# Universidad Complutense de Madrid
# Facultad de Informática


Sistemas Informáticos 2010/2011

# RSA@Cloud
# Criptoanálisis eficiente en la Nube

Autores:
Alberto Megía Negrillo
Antonio Molinera Lamas
José Antonio Rueda Sánchez

Directores de proyecto:
José Luis Vázquez-Poletti
Ignacio Martin Llorente
Dpto. Arquitectura de Computadores

*Alberto Megía Negrillo* *Antonio Molinera Lamas* *José Antonio Rueda Sánchez*

# Agradecimientos

*Un trabajo como este no habría sido posible sin la cooperación de mucha gente. En primer lugar queremos agradecérselo a nuestras familias que tanto han confiado y nos han dado. A nuestros directores José Luis Vázquez-Poletti e Ignacio Martín Llorente, por conseguir llevar adelante este proyecto. Por último, a todas y cada una de las personas que forman parte de la Facultad de Informática de la Universidad Complutense de Madrid, nuestro hogar durante estos años.*

# Resumen

En el presente documento se describe un sistema que aprovecha las virtudes de la computación Cloud y el paralelismo para la factorización de números grandes, base de la seguridad del criptosistema RSA, mediante el empleo de diferentes algoritmos matemáticos como la división por tentativa y criba cuadrática. Se ha optado por el uso de una infraestructura de Cloud público de Amazon y servidores de red propios, que permiten alcanzar un resultado óptimo en términos de tiempo y coste.

Una de las características fundamentales del proyecto viene dada por el diseño del sistema en módulos, comenzando por el software de simulación "*Forecaster"*, el cual se encarga de estimar el tiempo de computación necesario y el coste sobre la infraestructura Cloud de Amazon. Para la factorización de una clave RSA empleando paralelismo se ha creado otro módulo, "*Engine*", destinado a conectar una red de servidores y a generar y distribuir las operaciones de cálculo sin la necesidad de interacción por parte del usuario. Así mismo se ha incorporado al sistema un módulo de representación gráfica, "*Codeswarm*", con el fin de visualizar las interacciones entre las distintas máquinas.

**Palabras clave:** Cloud, RSA, criptoanálisis, paralelismo, Amazon, Codeswarm, Criba Cuadrática, factorización, SSH, Perl.

# Abstract

This paper describes a system which takes the most advantage of Cloud computing and parallel programming in order to factorize very big integers, the real basis of RSA cryptosystem's security, by executing different mathematic algorithms like trial division and quadratic sieve. The combination of the Amazon's public Cloud infrastructure and private servers has been chosen to achieve this goal, since both make possible to reach optimum results in terms of time and cost.

One of the most relevant characteristics of this programming project is that the system has been designed into modules, starting with the simulation software "*Forecaster*", whose purpose is to estimate the required processing time and the cost on the Amazon's Cloud infrastructure. Another module called "*Engine*" has been developed to factorize RSA keys thanks to parallel programming properties. Its aim is to connect a server network and to generate and assign computer operations without any interaction by the user. Furthermore, a graphic representation application, "*Codeswarm*", has been included as a module into the system, to visualize interactions between servers that carry out with the factorization task.

**Keywords:** Cloud, RSA, cryptanalysis, parallel programming, Amazon, Codeswarm, Quadratic Sieve, factorization, SSH, Perl.

# 1. Computación Cloud

## 1.1 Introducción

Nuestra aplicación está dirigida para ser ejecutada en la Nube o Cloud. Dado que a priori desconocemos los medios y el tiempo necesarios para realizar dentro de un presupuesto concreto la tarea deseada, la Computación Cloud se antoja como la solución más adecuada. En capítulos posteriores se dará mayor información acerca de la variabilidad del coste y tiempo de ejecución de algoritmos de factorización de números enteros en función de la longitud de la clave, algoritmo elegido o recursos empleados. Dicho esto, la mejor forma de justificar esta elección es realizar una breve visión general sobre esta tecnología.

Resulta complejo hacer una definición breve del concepto de "Cloud" (o "la Nube" en castellano). Si queremos emplear una definición oficial y objetiva de este término, nada mejor que emplear la que nos aporta el Instituto Nacional de Estándares y Tecnología (NIST), una agencia de la Administración de Tecnología del Departamento de Comercio de EE. UU.:

"*La Computación Cloud es un modelo para habilitar acceso conveniente por demanda a un conjunto compartido de recursos computacionales configurables, por ejemplo, redes, servidores, almacenamiento, aplicaciones y servicios, que pueden ser rápidamente aprovisionados y liberados con un esfuerzo mínimo de administración o de interacción con el proveedor de servicios*"*[MELL] [TAYL]*.

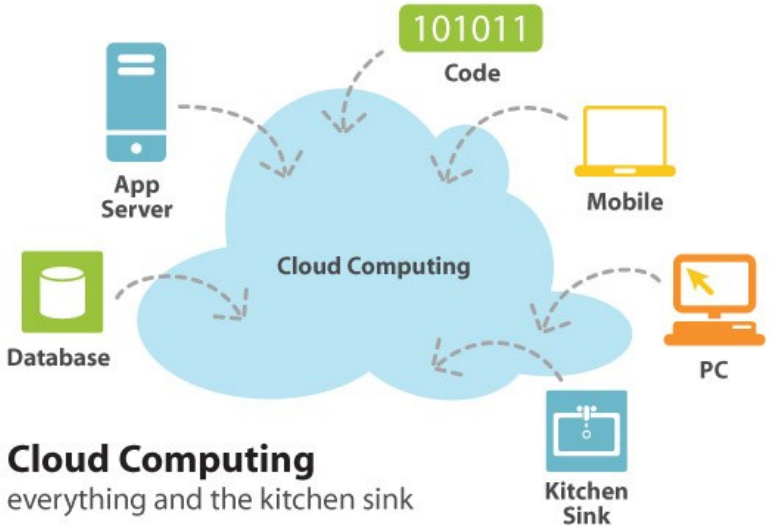


Figura 1.1: El modelo Cloud. Uno de los objetivos principales del Cloud es ofrecer servicios informáticos de cualquier índole independientemente del dispositivo que estemos empleando o dónde se encuentren los recursos a los que queremos acceder.

Acercándonos a este término de una manera menos general y más concreta, podemos describir la Computación Cloud como una computación independiente de la localización del usuario, en la que servidores compartidos proveen recursos, software y/o datos en función de la demanda deseada en cada momento (escalabilidad), sin que el usuario tenga la necesidad de tener conocimientos acerca de los servicios que le son proporcionados [SCHU]. Por tanto, se presenta como la evolución natural de la creciente expansión del empleo de la virtualización, la computación orientada a servicios y el concepto de ésta como un servicio público más, como puedan ser el agua y la electricidad, entre otros.

Este nuevo modelo significa la industrialización de la computación [BUYY]. Por cuestiones económicas y de agilidad, es una clara alternativa a los centros de datos para las compañías que requieran servicios. Éstos últimos siempre han permitido añadir o liberar recursos, pero nunca se ha podido hacer de una manera automatizada y "a la carta" en función de las necesidades y circunstancias.

## 1.2 Características

De forma generalizada, una nube reúne ciertas características por las cuales esta nueva forma de computación supone una importante novedad que puede influir notablemente en la industria durante los próximos años, a nivel empresarial y de investigación (Amazon EC2, Microsoft Cloud, IBM, Google Docs) y dentro del ámbito social (Facebook, Twitter, Tuenti). Estas cualidades son:

**- Escalabilidad y elasticidad "*self-service*"**: Los recursos deben estar siempre disponibles, por lo que una nube debe estar diseñada para que éstos puedan ser ampliados y reducidos según los diferentes periodos de demanda de una forma ajustada. Los clientes podrán elegir diversas opciones de SO y potencia de cómputo, cuyo rendimiento será monitorizado para garantizar el servicio ("*metering*"): no se pagan servicios que no se utilizan.

**- Acceso deslocalizado**: Dado que se accede a los recursos Cloud a través de la red mediante virtualización, es posible conectarse a través de cualquier PC o dispositivo móvil con acceso a Internet.

- **Interfaz de programación de aplicaciones (API) estandarizada**: un servicio Cloud debe ser capaz de conectar su software con las aplicaciones del cliente a un nivel comparable al de una interfaz de usuario de un PC.

**- Capacidad de recuperación ante fallos**: Si un nodo de la red Cloud falla, dicha incidencia no afecta a la disponibilidad de los recursos, luego el único punto débil es la red en sí. El desarrollo de las infraestructuras y la topología de la red hacen cada vez más eficiente esta tecnología.

**- Seguridad:** El proveedor centraliza la información e incrementa la inversión sobre la seguridad de los recursos para evitar que otros clientes o hackers tengan acceso a ellos. La distribución de los datos en numerosos servidores y la capacidad de desviar

recursos propios de esta tecnología aumentan la seguridad en comparación con los centros de datos tradicionales [DUMM].


Figura 1.2: La versatilidad de la Nube. Entre las cualidades de los servicios Cloud como Amazon EC2 encontramos el acceso remoto a nivel mundial, monitorización en tiempo real y escalabilidad de los recursos contratados a un nivel nunca visto hasta ahora e imposible para los servicios de "*hosting*" tradicionales. [ARMB]

# 1.3 Perspectiva histórica

La computación Cloud es un servicio. Esta afirmación se ha comentado y se va a repetir varias veces a lo largo de estas páginas. Puede parecer muy reiterativo, pero este proyecto se ha realizado en parte con la gran convicción de que, si aún no lo es, el concepto de 'nube' como servicio generalizado será un hecho. Todavía gran parte de la sociedad lo desconoce por completo y sólo una parte de ella, más vinculada a las nuevas tecnologías, lo emplea como tal.

Para argumentar este punto de vista, nada mejor que realizar un recorrido histórico desde los albores de la civilización hasta nuestro días durante el cual descubriremos algunos ejemplos de avances tecnológicos que, tras su hallazgo, el ser humano consideró adaptarlos como un servicio a toda la comunidad en favor de su bienestar y progreso.

Comenzamos con el origen de la agricultura, cuyo comienzo se encuentra en el período Neolítico, cuando la economía de las sociedades humanas (clanes en aquella época) evolucionó desde la recolección, la caza y la pesca a la agricultura y la ganadería. Estas nuevas actividades permitieron mayor densidad de población por la disponibilidad de alimento para un mayor número de individuos. Es decir, con mayor o menor eficacia y disponibilidad, se convirtieron en un servicio de sustento alimenticio. En la Era Moderna, con la agricultura las sociedades se tornan cada vez más sedentarias y aparece el concepto moderno de la propiedad sobre los bienes inmuebles, se amplía la división del trabajo y surge una sociedad más compleja con actividades artesanales y comerciales especializadas, los asentamientos agrícolas y los conflictos por la

interpretación de linderos de propiedad dan origen a los primeros sistema jurídicos y gubernamentales. Estas circunstancias dan lugar a nuevos servicio básicos como la vivienda o el agua.

### 1.3.1 La electricidad, un ejemplo antecedente del Cloud

Del mismo modo que el vapor fue el combustible que hizo posible la Revolución Industrial entre la segunda mitad del siglo XVIII y principios del XIX, la electricidad lo fue para definir la forma en que crecería el siglo XX. A finales del siglo XIX, automatizar ciertos procesos industriales de construcción a través de la electricidad se convirtió en la clave para el éxito de un negocio. Sin embargo, por aquel entonces las empresas que optaban por esta vía debían invertir en construir, mantener y evolucionar su propia generación de electricidad. Esto implicaba el contar con equipos especializados de electricistas y mecánicos. Dichos equipos podían formar parte de la propia compañía, si ésta tenía los medios suficientes.

En 1888 Nikola Tesla, el padre del sistema eléctrico que hoy disfrutamos, patenta un nuevo motor de inducción que dos años después le lleva a patentar el generador de corriente alterna. Este descubrimiento permite, entre otras cosas, liberar a la industria de las cadenas que supone la limitación del tamaño de la instalación que, con la corriente continua, no permitía sobrepasar los 400 metros.

Los derechos de éstas y otras patentes relacionadas con sus sistemas de corriente alterna, transformadores, motores y generadores, los vendió a George Westinghouse, fundador de la Westinghouse Company, pionera en el desarrollo comercial de la corriente alterna. En 1893, durante una feria en Chicago, Westinghouse y Tesla presentaron todo un sistema eléctrico de corriente alterna a escala a fin de demostrar sus bondades. En 1895, Westinghouse pone en servicio la primera planta de generación de electricidad comercial en corriente alterna, La Planta del Niágara. Nacía la primera central que permitía el acceso externo al servicio eléctrico.

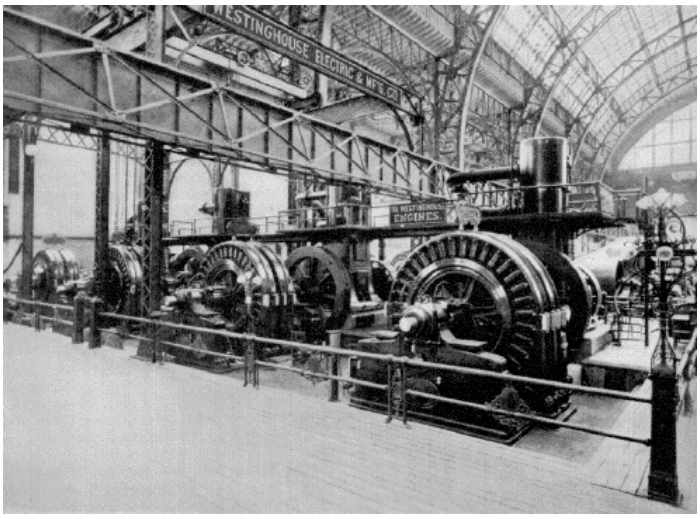


Figura 1.3: Generadores eléctricos de la Westinghouse Company. El despliegue de la infraestructura de corriente alterna en las ciudades supuso la generación de energía eléctrica barata y habilitó la creación de aparatos eléctricos domésticos fuera del ámbito industrial.

Las primeras centrales eléctricas se instalaron dentro de las ciudades, y quemaban carbón o petróleo para generar vapor; el vapor, al circular a presión por unas turbinas generadoras de corriente alterna, permitían obtener la electricidad, y esta electricidad era distribuida a las industrias que la necesitaban, siendo posteriormente introducida de forma paulatina en los hogares. Con el tiempo, las centrales eléctricas hubieron de ser situadas fuera de las ciudades porque con el incremento de la demanda, estas instalaciones crecieron de tamaño hasta convertirse en los grandes complejos generadores actuales [MART].

Gracias a los trabajos de Tesla sobre el transporte de la energía eléctrica, hoy podemos encender la luz de nuestras casas sin preocuparnos sobre si la energía procede de una central térmica, solar o nuclear. La electricidad es un ejemplo (entre muchos otros) de tecnología cuya evolución se dirigió de forma vertiginosa a un enfoque en el cual la ciudadanía se beneficiase de ella.

La agricultura, el agua, el vapor, la electricidad y más ejemplos de este tipo como la televisión, la telefonía o el mismo Internet son conceptos de servicio básico a lo largo de la historia. Ahora muchas compañías de TI se esfuerzan para ser proveedores públicos de computación.

La evolución de la informática hoy en día estriba en que se está pasando de un modelo donde cada persona o empresa hace uso de la informática y el procesamiento de datos como algo privado, localizado y a veces limitado, a otro donde ya no necesitamos saber dónde se encuentra almacenado el documento que estamos leyendo con una aplicación que no está instalada en nuestro ordenador.

Dicho de otro modo, estamos pasando de un modelo donde cada entidad es responsable de su infraestructura TI a un modelo donde el servicio básico (como lo es la alimentación o la electricidad) puede ser gestionado de una forma global desde y para la sociedad. Esta idea proviene de Nicolas Carr, que observó en su libro "*The Big Switch*" cómo la infraestructura de los servicios de cómputo empezaba a ser comparable con el desarrollo de la electricidad como utilidad [CARR].

### 1.3.2 Evolución de la Computación Cloud

El concepto básico de la computación en nube se remonta a la década de 1960 cuando John McCarthy, famoso científico e informático ganador del Premio Turing en 1971, expresó que "algún día la computación podrá ser organizada como un servicio público". Casi todas las características modernas de la computación en nube (elasticidad del suministro, caracterizado como un servicio público o la ilusión de oferta ilimitada), la comparación con la industria eléctrica y su uso de forma pública, privada, gubernamental y comunitaria se exploró a fondo en "*The Challenge of the Computer Utility*" (Douglas Parkly, 1966). También surgió la idea de una "red de ordenadores intergaláctica" por parte de JCR Licklider, quien era responsable de permitir el desarrollo de ARPANET ("Advanced Research Projects Agency Network") en 1969 [MOHA].

En los años 80, se pasa de los "*mainframes"* centralizados de las grandes organizaciones a la creación del PC de IBM durante los años 80, lo que implica en una descentralización y personalización de la informática y el primer paso para extenderla a toda la sociedad (gracias también al SO Windows de Microsoft y los productos de su competidora Apple). En las siguientes dos décadas se crea y perfecciona el modelo cliente-servidor (SOA) potenciado por el nacimiento y la expansión de Internet o Red de Redes (la cual es ya conocida por todos y de la cual se podrían escribir cientos de páginas), que se convertirá en la infraestructura básica e imprescindible para el posterior desarrollo de la Nube.

El término 'nube' fue prestado de la telefonía cuando las empresas de telecomunicaciones, que hasta la década de 1990 principalmente ofrecían circuitos de datos punto a punto, empezaron a ofrecer redes privadas virtuales (VPN) con una calidad comparable de servicio pero con un coste de infraestructura y mantenimiento mucho menor. Al poder desplazar el tráfico de información para equilibrar el uso de la red, eran capaces de utilizar el ancho de banda total disponible de la red con mayor eficacia. El concepto de 'nube' se utilizaba para denotar el punto de demarcación que permitía diferenciar la parte de la red que era responsabilidad del proveedor de la que era propiedad del usuario. La computación en nube emplazará este límite al conjunto de servidores y la infraestructura de red necesarios para el suministro de este servicio con respecto al hardware del cliente. El primer uso académico de la expresión "computación en nube", fue durante una conferencia de Ramnath Chellappa (actual profesor de la Emory University) en 1997.

Amazon desempeñó un papel clave en el desarrollo de la computación en nube mediante la modernización de sus centros de datos tras "la burbuja de las puntocom", la cual, como muchas otras redes de computación, aprovechaban tan sólo un 10% de su capacidad normalmente salvo durante picos ocasionales. Habiendo constatado que esta arquitectura dio como resultado importantes mejoras en la eficiencia interna de la compañía, Amazon inició un programa de desarrollo de productos nuevos para ofrecer computación Cloud a clientes externos, y lanzó Amazon Web Service (AWS) en 2006. [RAGH]

Este modelo arquitectónico fue inmortalizado por George Gilder en su artículo de octubre de ese mismo año en la revista "Wired" titulado "Las Fábricas de Información".

Las granjas de servidores acerca de las cuales Gilder escribió eran similares en su arquitectura al cómputo grid, pero mientras que los grids son utilizados para aplicaciones de cómputo técnico "loosely coupled" (o sea un sistema compuesto de subsistemas con cierta autonomía de acción a la par que mantienen una interrelación continua con los otros componentes) este nuevo modelo de nube se estaba aplicando a los servicios de Internet.

En 2007, Google, IBM y un gran número de universidades se embarcó en un proyecto a gran escala de investigación sobre computación Cloud. A principios de 2008, Eucalyptus[1], una API compatible con la plataforma AWS, se convirtió en el primer software libre para el despliegue de nubes privadas. En 2008 OpenNebula[2], software cuyo desarrollo fue incluido dentro del proyecto RESERVOIR financiado por Comisión Europea, se convirtió en el primer software de código abierto para la implementación de nubes privadas e híbridas y para la federación de nubes. En el mismo año, los esfuerzos se enfocaron en proporcionar garantías de calidad de servicio (como es requerido por las aplicaciones interactivas en tiempo real) para infraestructuras basadas en Cloud, en el marco del proyecto financiado por la Comisión Europea IRMOS[3].

A mediados de ese año el prestigioso Grupo Gartner (consultora con proyectos de investigación de TI y de firma consultiva con sede en Stamford, EE. UU) vio en la computación Cloud una oportunidad "para dar forma a la relación entre los consumidores de servicios TI, los que utilizan los servicios de TI y los que los proveen" y observó que " la migración de los proyectos de las grandes organizaciones a modelos de servicio por tiempo de uso se traducirá en un crecimiento espectacular en los productos de TI en algunas áreas y una reducción significativa en otras", resaltando así la repercusión que tienen y tendrán las tecnologías basadas en la Nube. En junio de 2010, Gartner publicó un informe en el que confirmaba el crecimiento de la computación en la Nube. Sólo los servicios vinculados a estas tecnologías registraron unos ingresos de 68.300 millones de dólares (unos 55.700 millones de euros) en todo el mundo en el año 2010; la cifra supone un incremento del 16,6% respecto al año anterior. El informe estima también que este sector moverá 148.000 millones de dólares en 2014. [CINC]

[1] http://www.eucalyptus.com/
[2] http://opennebula.org/
[3] http://www.irmosproject.eu/

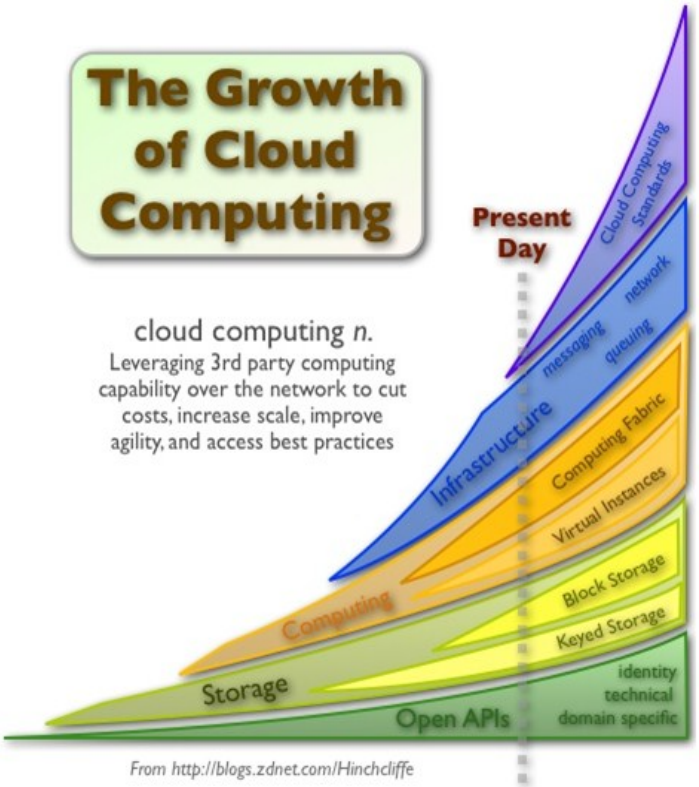


Figura 1.4: El auge de la computación Cloud. Un indicador del avance imparable de esta tecnología es el estudio de la consultora IDC que asegura que la computación Cloud estimulará un crecimiento aproximado de $1 trillón en la facturación consolidada de las empresas sólo de EEUU para el año 2014.

Desde el punto de vista de los proveedores informáticos (*hardware* y *software*), todas o casi todas las grandes empresas del sector han lanzado estrategias para toda la década: IBM, Microsoft, Oracle, Hewlett-Packard, Cisco, EMC, etc. Lo mismo se puede decir de las operadoras de telecomunicaciones europeas (Telefónica, Vodafone, France Telecom, Deutch Telecom) o americanas (Verizon, ATT). A todas ellas se unen las empresas, por excelencia, de Internet que ya son, *per se*, empresas de la Nube: Google, Yahoo!, Amazon o las redes sociales, tales como Facebook, Twitter o Tuenti.

¿Qué sucede en el resto de empresas y organizaciones no específicas de informática o telecomunicaciones? Se puede preguntar alguno que no quiera admitir que la Nube repercute a todos los niveles de la sociedad. Pues que tanto las grandes empresas como las pequeñas y medianas se están posicionando y migrando, gradualmente, a la Nube. En España, por ejemplo, podemos citar un caso paradigmático, Ferrovial, la gran empresa multinacional de infraestructuras. En diciembre de 2009 firmó un acuerdo con Microsoft para sustituir en los siguientes años sus servicios informáticos por servicios en la Nube de la multinacional estadounidense, y en el mes de junio de 2010 ha extendido su paso a la Nube firmando un acuerdo de "*outsourcing*" con Hewlett-Packard que complementa la estrategia de Ferrovial de posicionamiento en la Nube para los próximos años.


Figura 1.5: La competencia por la cuota de mercado es muy grande. Actualmente Amazon Web Services es líder en beneficios, seguido de Google e IBM en 2º y 3er lugar respectivamente. Les siguen Windows Azure (4º), Salesforce.com (5º), Oracle y Sun (6º), EMC y Vmware (7º), HP (8º), SAP (9º) y Dell (10º) [MAGU].

En estos momentos muchos sectores de la población está utilizando la Nube cuando enviamos un correo electrónico por Gmail, Yahoo! O Hotmail, escuchamos música en Spotify (el innovador servicio sueco de "*streaming audio*"), es posible subir y descargar fotografías y video en Flickr o Youtube, consultamos nuestra posición geográfica en Google Maps, en nuestro teléfono "*smartphone*", o utilizamos la reciente aplicación Places de la red social Facebook para aplicaciones de geolocalización. Todo ello unido al uso de almacenamiento masivo en la Red cada vez que utilizamos información en esos servicios. Las nubes de servidores han favorecido que el correo electrónico pueda ser leído y archivado a distancia en Google Mail, Yahoo! Mail, Microsoft Mail, etc. También en el campo profesional de la gestión empresarial, se puede utilizar un programa de software de CRM (gestión de relaciones con los clientes) mediante una tasa fija en el sitio de Salesforce.com.

También es destacable mencionar las grandes innovaciones tecnológicas que vienen asociadas a la Nube y que producirán un cambio social, además del cambio tecnológico, difícil de predecir: la Web en tiempo real, la mencionada geolocalización, la realidad aumentada, y la pronta llegada de la telefonía móvil LTE de cuarta generación (4G), unida a los nuevos estándares de USB, Bluetooth e implantación de redes inalámbricas WiFi y WiMax.

### 1.3.3 La alternativa que engloba los paradigmas de la computación distribuida

Como se ha mencionado anteriormente, no es hasta bien entrados los años 90 cuando el rumbo de la computación de altas prestaciones cambia de un modelo centralizado a

otro en el que intervienen un número indeterminado de sistemas para desempeñar una tarea conjunta. En realidad, esta tendencia no fue el producto de una gran revolución tecnológica, sino que es el resultado de la combinación del modelo cliente-servidor y el protocolo TCP/IP, junto con sistemas de almacenamiento apropiados para la arquitectura distribuida y de autenticidad y seguridad para el envío de información por la red.

El primer modelo de computación distribuida fue el conocido como **Clúster**. Este término se aplica a los conjuntos o conglomerados de sistemas construidos mediante la utilización de componentes de *hardware* comunes, unidos mediante una red de alta velocidad, que se comportan como si fuesen una sola máquina. Estos sistemas presentan un alto rendimiento y disponibilidad, balanceo de carga y escalabilidad, a pesar de contar con inconvenientes como su mayor dificultad de mantenimiento y la necesidad de un paradigma de programación basado en memoria distribuida.

El modelo denominado "***Intranet Computing***" pasó a emplear ciertos componentes de los clúster para organizar un determinado conjunto heterogéneo de equipos informáticos para no desperdiciar la potencia computacional distribuida no aprovechada, suponiendo que los recursos no son dedicados y, por lo tanto, no siempre están disponibles. Con un coste casi nulo, es la mejor alternativa para aprovechar el procesamiento libre, sobre todo si se ejecutan trabajos independientes.

"***Internet Computing***" supone la extensión del anterior modelo a la Red de redes, por lo que la seguridad y el ancho de banda de la red son los factores más importantes de esta alternativa. Aunque pueda parecer muy similar a la computación Cloud, el hecho de trabajar con máquinas físicas y de diferentes propietarios de cualquier parte del mundo acarreaba problemas derivados de la disparidad en políticas de seguridad y procedimientos de explotación de recursos, entre otros.

La computación "*Peer-to-Peer*" (**P2P**) supone una novedad al no basarse en el modelo cliente-servidor, en el cual un cliente distribuye tareas a distintos servidores. La propuesta de P2P para evitar los cuellos de botella del modelo cliente-servidor es un sistema que permite la comunicación entre servidores: actúan simultáneamente como clientes y servidores respecto a los demás nodos de la red. Este modelo es muy común en la compartición de ficheros en entornos distribuidos y en la telefonía de voz IP (voIP).

Ninguna de las herramientas anteriores permite compartir recursos distribuidos en diferentes dominios de administración, cada uno con sus propias políticas de seguridad y gestión de recursos. La **computación Grid** hace esto posible, incluyendo compartir además tanto almacenamiento como aplicaciones además de capacidad de procesamiento. Como para poder utilizar recursos de diferentes dominios de administración una organización debía unirse a esa red, y por tanto compartir sus recursos propios con el resto de organizaciones integrantes, su aplicación presenta un rendimiento muy alto y es por tanto muy frecuente en la computación científica [POLE].

La **computación Cloud** es el paradigma actual que mejor reúne todos los aspectos y ventajas de los modelos anteriormente descritos. Esta nueva tecnología, gracias al desarrollo de la virtualización, la arquitectura de red orientada a servicios e Internet, permite un alto grado de rendimiento y disponibilidad, balanceo de carga y escalabilidad así como elasticidad, lo que permite personalizar la cantidad de recursos

disponibles rápidamente, todo ello sin tener que invertir en una gran infraestructura de computación si no se desea, ni tener que compartirla con otras organizaciones, como ocurría por ejemplo con la computación Grid. Aunque esta tecnología dependa de la banda ancha disponible en Internet, ésta aumenta cada día, y en cuanto a la infraestructura requerida y la seguridad en el envío de información, están asegurados si se opta por una nube pública, ya que los proveedores de servicios hacen del Cloud su modelo de negocio, dando una sensación de acceso por demanda a recursos de computación infinitos. Uniendo a estas ventajas un rendimiento medio mayor y la portabilidad de información que ofrece la virtualización, o el ahorro que supone la posibilidad de no abordar los gastos de mantenimiento y actualización, entre otras, podemos afirmar que el modelo Cloud permite cubrir todos los aspectos de los anteriores modelos con una relación Coste/Rendimiento mucho mejor.

## 1.4 Elementos / Conceptos constituyentes

El servicio de la tecnología de la computación Cloud requiere una combinación de *hardware* y *software* encaminada a suministrar un servicio a un número considerable de usuarios dispuestos a pagar por él. Dependiendo del servicio específico proporcionado, algunos elementos constituyentes son configurados por el proveedor del servicio o se dejan a disposición de las necesidades del cliente: esta elección es dependiente de las diferentes capas de servicio (infraestructura, plataforma y software, principalmente) que serán descritas en un apartado posterior. Los dos componentes más significativos de la arquitectura de computación Cloud se conocen como el "*front-end*" y el "*back-end*". El frontal es la parte vista por el cliente, es decir, el usuario de la computadora. Esto incluye la red del cliente (o computadora) y las aplicaciones utilizadas para el acceso a la Nube a través de una interfaz de usuario, como un navegador web. El extremo posterior de la arquitectura Cloud es la Nube en sí, que comprende computadoras, servidores y dispositivos de almacenamiento de datos.

Se necesitan importantes recursos físicos para tener capacidad de cómputo y almacenamiento así como una red que encamine la información que estas máquinas procesan o contienen hacia las terminales de los usuarios, sin importar dónde se encuentren. También hace falta *software* que emplee estos recursos materiales para dotar a cada cliente de su propia máquina remota simulada configurada a su elección. Para ello, a bajo nivel debe haber una capa que gestione el hardware que otorgue estos servicios personalizados a cada usuario en tiempo real: a este concepto se le llama virtualización. Este software simula la existencia de un conjunto de máquinas virtuales dentro de una misma máquina física y a la vez monitoriza su estado a través de un hipervisor, asigna recursos y establece prioridades a cada una de ellas. De esta forma se pueden generar plataformas para cada máquina en particular: a un software de virtualización se le añade una pila básica con un sistema operativo determinado, una conexión con la máquina a un servidor mediante redes virtuales y se despliega un ambiente de desarrollo de *software* específico. En su lugar, el proveedor puede establecer configuraciones determinadas de estos parámetros que sean suficientes para ofrecer al consumidor una aplicación en particular (*software* como servicio).

En resumen, los entornos de computación Cloud son diseñados para soportar múltiples usuarios independientes que no son propietarios de la máquina pero sí emplean una fracción de ella para su trabajo.

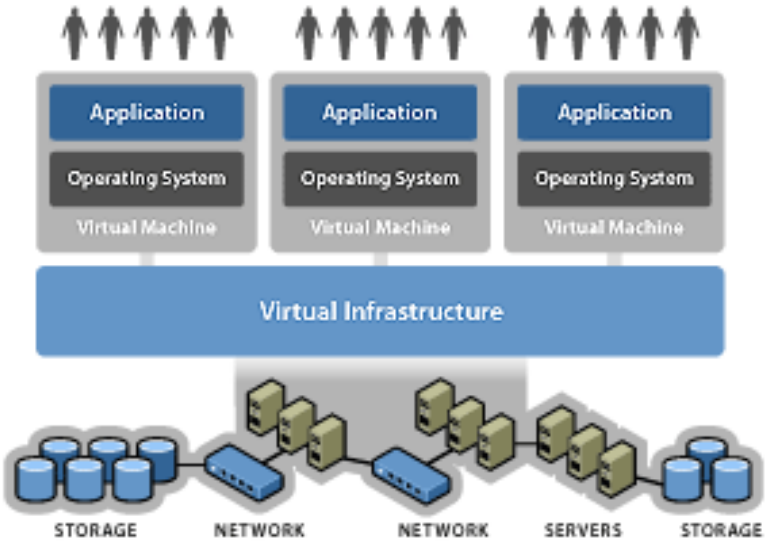


Figura 1.6: Esquema de una red virtualizada.

Con una infraestructura hardware típica de un CPD como primera capa e instalando sobre ella un hipervisor, se pueden crear un gran número de máquinas virtuales sobre el mismo, cada una con recursos asignados específicos. Tanto éstos como el SO y las aplicaciones se dejan a elección del usuario final[4].

Por poner un ejemplo, el enfoque típico de la infraestructura de centros de datos para el Cloud (Amazon EC2) es construir una colección de servidores de propósito general comerciales conectados por red estándar de área local (LAN) mediante tecnologías de conmutación. En la capa superior, el modelo de infraestructura como servicio (IaaS) se construye donde los usuarios tienen acceso a los recursos del sistema, mediante servidores y redes virtuales. Una plataforma de software de virtualización como Xen[5] es ampliamente considerada como el factor clave para IaaS, ya que proporciona a los usuarios un entorno de software estándar que conocen, además de un control "*multi-tenancy*" o de múltiples clientes a los administradores del servicio.

Tras esta introducción, en la que se hecho una descripción general de los diferentes elementos o componentes tecnológicos que hacen posible construir una nube independientemente del servicio que ofrezcan a sus clientes (o bien sea de carácter privado), se hará un recorrido de bajo a alto nivel aislando cada uno ellos para una mejor comprensión de los mismos. Los elementos constituyentes de la computación Cloud son: infraestructura, plataforma, aplicaciones y servicio y API's· [SAAV].

[4]http://www.vmware.com/es/virtualization/virtual-infrastructure.html
[5]http://www.xen.org/

### 1.4.1 Infraestructura virtualizada

La capa física de la computación Cloud es básicamente la misma que se ha empleado en informática desde sus inicios: un centro de datos (CPD), un supercomputador o un "*mainframe"*. Ahora bien, la arquitectura del mismo puede diferir para adaptarse a la virtualización, una delgada capa de software de nivel bajo que gestiona sus recursos. El objetivo es principalmente la maximización del rendimiento, que al fin y al cabo es el objetivo de cualquier avance en computación, y en este aspecto el Cloud no va a ser menos. De esta manera un proveedor de Cloud puede ofrecer servicios a miles o incluso millones de usuarios simultáneamente con un número de máquinas proporcionalmente mucho menor.

Figura 1.7: Pecera de un CPD.

Un centro de cómputo, centro de procesamiento de datos o centro de datos se encarga del procesamiento de información de forma sistematizada. Por lo general, estas computadoras se encuentran interconectadas en red y cuentan con conexión a Internet. Generalmente, todos los grandes servidores se suelen concentrar en una sala denominada "sala fría", "nevera", o "pecera". Esta sala requiere un sistema específico de refrigeración para mantener una temperatura baja, necesaria para evitar averías en las computadoras a causa del sobrecalentamiento y medidas contra incendios como sistemas de extinción por medio de agentes gaseosos.

**Infraestructura hardware**

Un centro de datos diseñado para computación Cloud suelen tomar la forma de centros de datos de nivel Tier 3 ó nivel Tier 4 según el estándar de sostenibilidad operativa del Uptime Institute (consorcio fundado en 1993, pionero en la divulgación de conocimiento en lo relativo a los centros de datos y autoridad a nivel mundial en este

campo) cuyo objetivo es medir el nivel de fiabilidad del centro de datos en base a los procesos y políticas operacionales, así como al tipo de construcción y ubicación del sitio[6].

Las alturas de una sala de un centro de datos o CPD son tres:

**Falso suelo:** debajo del suelo hay un nivel inferior contiene el cableado eléctrico junto con la salida del sistema de aire acondicionado. Suele medir como mínimo 40cm de alto, llegando a alcanzar los 1.50m en algunos casos.

**Superficie**: Dentro de la sala se disponen filas de bastidores o "*racks*" (armarios con estantes para máquinas), cuya altura estándar es 2,20m o 47U. Cada máquina suele medir 19 pulgadas de ancho, 90 cm de profundidad y una altura en cantidades enteras de la unidad de medida U. 1U corresponde a 1,75 pulgadas o 4,5 cm aproximadamente.

**Parte aérea**: Una estructura metálica en la zona alta de la sala por la cual residen los cables de datos y que tiene al menos 50 cm de altura.

El aislamiento térmico lo componen sistemas de aire acondicionado a presión en el suelo. Existen unas rejillas en la parte delantera de los "*racks*" por los que pasa el aire, que proviene de los sistemas de ventilación colocados a ras del suelo. Este sistema y la disposición en filas de los "*racks*" es el sistema típico en los CPD y deriva en la existencia de pasillos "fríos" y "calientes", los cuales evitan que las corrientes de aire a diferente temperatura se entremezclen.

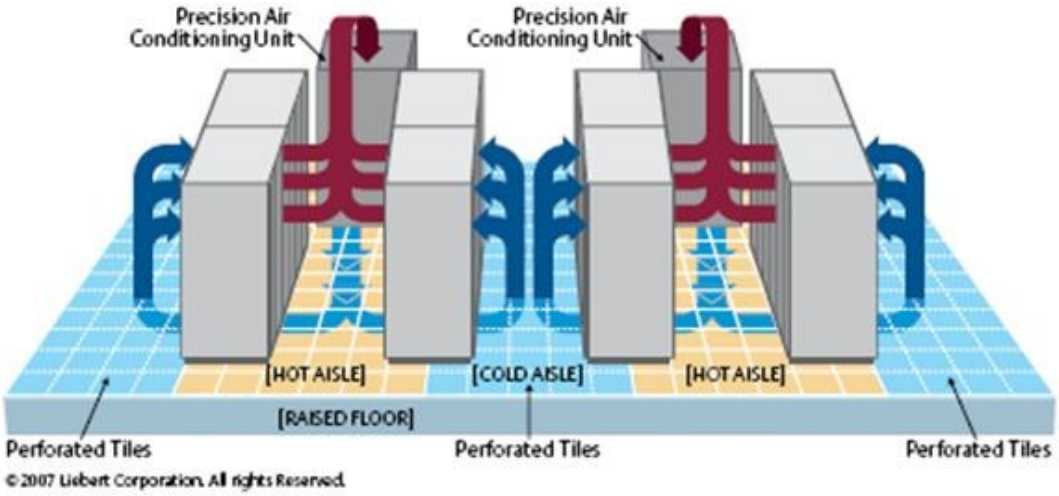


Figura 1.8: Sistema de refrigeración de un CPD.

La Figura 1.8 muestra el esquema de un sistema básico para el mantenimiento de la temperatura de las máquinas en condiciones óptimas. Las condiciones óptimas normalmente son: 22ºC, 2200 Kfrigocal/m² y 45-50% de humedad relativa.

El sistema anti-incendios consiste en la ubicación de tanques de gas supresor del oxígeno no corrosivo con alarmas de preaviso para el desalojo. La seguridad del recinto pasa por instalar controles de acceso con tarjeta electrónica y registros electrónicos del

[6]Services - Tier Certification: http://professionalservices.uptimeinstitute.com/tier_class.htm

personal que ha accedido a las instalaciones. En el sistema de suministro eléctrico predomina la redundancia de dispositivos, como se verá a continuación.

Un complejo de nivel 4 debe cumplir con una serie de requisitos[7].Esencialmente son los siguientes:

- Varios caminos independientes de distribución de conexión y de suministro eléctrico sirviendo a los equipos. Por lo general, una sola vía de distribución debe servir a los equipos en un momento dado.

- Todos los equipos tienen fuente de alimentación doble, la cual es totalmente compatible con la topología de la arquitectura del complejo.

- La instalación es totalmente tolerante a fallos, gracias al almacenamiento eléctrico y las redes de distribución, los cuales incluyen componentes redundantes.

- Todo el equipo de refrigeración es independiente y de doble encendido, incluidos los sistemas de HVAC ("Heating, Ventilating and Air Conditioning").

- La actividad planificada no alcanza la carga crítica máxima. El centro de datos puede sostener al menos un peor caso de eventos no planificados.

- El tiempo de inactividad anual medio es de 0,4 horas. Este tipo de centros de datos conllevan 15/20 meses para su construcción.

Un centro de datos suele contar con servidores (que realizan el trabajo de cómputo) o bien *mainframes*, discos de almacenamiento y tecnología de red para implementar un sistema virtualizado.

**Hipervisor**

En los servidores físicos normalmente se ejecuta un hipervisor o VMM ("Virtual Machine Monitor") que crea una fina capa de abstracción entre el *hardware* de la máquina física y el sistema operativo de las máquinas virtuales (imágenes de SO con una configuración *hardware* virtual de potencia y almacenamiento) y se encarga de crear, destruir, y gestionar los recursos (CPU, memoria, red, almacenamiento).

A estos sistemas operativos inquilino o máquinas virtuales se les asigna una parte de los recursos del servidor físico, de una manera en la que éstas no tienen conocimiento de ningún recurso físico con excepción de los que le sean asignados por el hipervisor. Existen dos tipos de hipervisor o VMM:

- **Anfitrión**: El hipervisor se ejecuta dentro de un entorno de sistema operativo convencional. Con la capa del hipervisor como un segundo nivel de software, los sistemas operativos invitados se lanzan en un tercer nivel por encima del *hardware*.

- **Nativo**: El hipervisor se ejecuta directamente en el *hardware* del *host* para controlar la memoria, procesador, discos y conexiones de red así como para monitorizar los sistemas operativos invitados. Un sistema operativo invitado por lo tanto se ejecuta en otro nivel por encima del hipervisor. Este modelo representa la aplicación clásica de las

[7]TIA-942 Data Center Standanrds Overview. ADC Communications: http://www.adc.com

arquitecturas de máquina virtual. El primer hipervisor fue CP/CMS, desarrollado por IBM en la década de 1960, antecesor del IBM z/VM.

Evidentemente, en la computación Cloud se emplea el tipo de hipervisor nativo por su mayor rendimiento, dado que éste trabaja directamente sobre la máquina física sin niveles intermedios. Los hipervisores anfitriones suelen ser de uso doméstico o académico.


Figura 1.9: Monitorización de una infraestructura virtualizada.

El hipervisor permite comprobar el rendimiento de cada máquina virtual bajo su control. La escalabilidad propia de la computación Cloud es posible gracias a la asignación dinámica de recursos (procesamiento, RAM, almacenamiento, ancho banda de red) desde una interfaz del hipervisor o monitor.

Las MV pueden ser totalmente virtualizadas, paravirtualizadas, o híbridas[8].

**Virtualización completa**

En un entorno totalmente virtualizado, la MV desconoce que el hardware no es estrictamente físico. El hipervisor en este caso debe traducir, mapear y convertir las peticiones del sistema inquilino en las solicitudes de recursos adecuadas en el servidor físico o host, lo cual genera de trabajo en la memoria física. Casi todos los sistemas pueden ser virtualizados utilizando este método, ya que no requiere ninguna modificación del sistema operativo, sin embargo, se requiere una CPU que soporte virtualización para la mayoría de hipervisores que realizan virtualización completa.

Dentro de la virtualización completa, tenemos los siguientes hipervisores, entre otros: Parallels Workstation, VirtualBox, Virtual Iron, Oracle VM, Virtual PC, Virtual Server, Hyper-V de Microsoft, Vmware Workstation, Vmware Server (antes GSX Server), QEMU, Adeos, Win4BSD, Win4Lin Pro y Egenera vBlade.

[8]http://itbully.com/articles/virtualization-paravirtualization-whatever-you-say

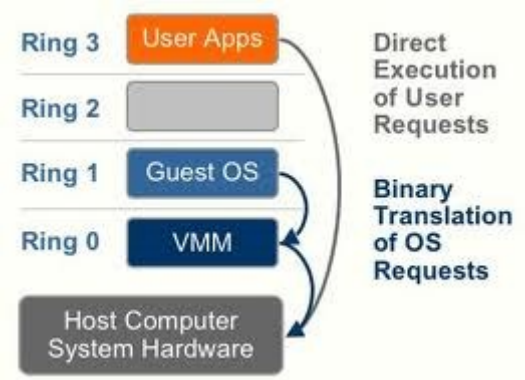


Figura 1.10: Virtualización completa.

Mientras que las peticiones del usuario son enviadas directamente al hardware para su ejecución, las peticiones del kernel del sistema operativo requieren una transformación binaria previa realizada por monitor de la máquina virtual.

**Virtualización parcial o paravirtualización**

En un entorno paravirtualizado, la MV es consciente del hipervisor y las interfaces que rigen los recursos del sistema anfitrión, en las que el sistema hipervisor lleva el control de acceso y la asignación de recursos en tiempo real. Esto se traduce en un rendimiento de la máquina física casi real ya que la MV considera el mismo hardware que el servidor anfitrión y por lo tanto se puede comunicar con él de forma nativa. Algunos sistemas de tipo Unix, como Linux, algunas variantes de BSD, Plan9, y OpenSolaris son conocidos por soportar este método de virtualización. Sin embargo, la instalación de sistemas operativos inquilinos paravirtualizados supone requerir más conocimientos sobre el sistema operativo con el fin de poder gestionar kernels y dispositivos especiales conscientes de la presencia de un hipervisor, el cual reemplaza las instrucciones del sistema operativo no virtualizables.

Algunos ejemplos de hipervisores de paravirtualización son Xen, Virtuozzo, Vserver, y OpenVZ (que es código abierto y la versión de desarrollo de Parallels Virtuozzo Containers).

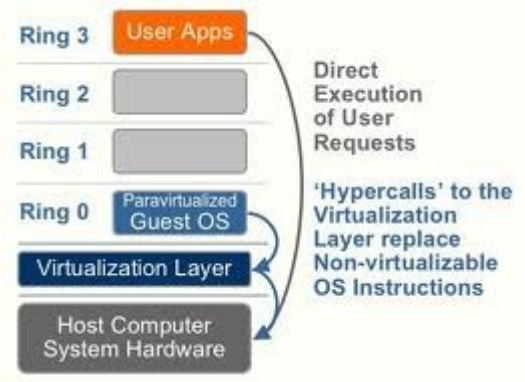


Figura 1.11: Paravirtualización.

La ventaja de rendimiento de la paravirtualización sobre la virtualización completa puede variar mucho dependiendo de la carga de trabajo a la que sea sometido el *hardware*. Como la paravirtualización no soporta sistemas operativos sin modificar, su compatibilidad y portabilidad son algo pobres.

**Virtualización híbrida**

En la virtualización híbrida o asistida por hardware, el hardware proporciona soporte arquitectónico que facilita la construcción de un monitor de máquina virtual y permite que los sistemas operativos invitados que se ejecuten de forma aislada, proporcinando así un entorno de virtualización completa.

Algunos de los hipervisores híbridos adaptados al hardware especializado en virtualización son Kernel-based Virtual Machine (KVM), Vmware Workstation, Vmware Fusion, Virtual PC de Microsoft, Xen, Parallels Desktop para Mac Server, Oracle VM para VirtualBox SPARC, y Parallels Workstation.

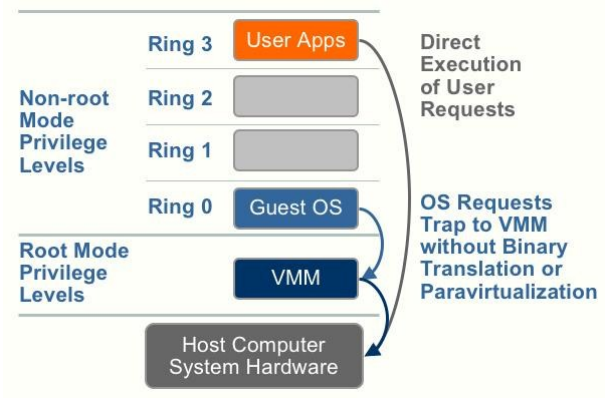


Figura 1.12: Virtualización asistida por hardware.

Los fabricantes de hardware están adoptando rápidamente la virtualización y el desarrollo de nuevas características para simplificarlas técnicas de virtualización. La primera generación incluye Intel Virtualization Technology yAMD-V, hardware que permite procesar llamadas del sistema operativo directamente.

## 1.4.2 Plataforma

Una plataforma se compone de un sistema operativo (el coordinador del sistema informático) que a su vez se basa en el conjunto de instrucciones específico para un procesador o microprocesador (el hardware que realiza las operaciones lógicas y gestiona el flujo de datos en el ordenador). El sistema operativo debe estar diseñado para funcionar con un determinado conjunto de instrucciones de una familia de procesadores en concreto. Actualmente los procesadores de los dos principales líderes en este sector, Intel y AMD, son compatibles con prácticamente la totalidad de los sistemas operativos. Hay también otros componentes implicados en cualquier plataforma informática, tales como la placa base y los buses de datos, pero estas piezas se han convertido cada vez más en partes modulares y estandarizadas compatibles con cualquier juego de instrucciones o sistema operativo.

Históricamente, la mayoría de los programas han sido escritos para ejecutarse en una plataforma en particular. Para cada plataforma se proporcionan APIs diferentes para los diferentes servicios del sistema o los paquetes de software conocidos como "*solution stacks*" o "*software bundle*" (WINS o WIMP para Windows, LAMP o LYME para

Linux)[9]. Por lo tanto, un programa de PC tendría que ser por escrito para funcionar en una plataforma con un sistema operativo Windows 2000 y procesador Intel y luego otra vez para hacer lo propio con un Apple y Mac OS X, uno de los motivos por los que esta compañía empezó a diseñar un sistema operativo compatible con familias Intel. A pesar de que estas diferencias de plataforma siguen existiendo y probablemente siempre habrá diferencias de sistemas propietario, están apareciendo nuevas interfaces estandarizadas de código abierto que permiten ejecutar diferentes programas en distintas plataformas o interactuar con diferentes plataformas a través de la mediación de programas denominados "29orcé29".

Los términos "*cross-platform*"[10], independiente de la plataforma o portable se utilizan con frecuencia para describir los sistemas operativos y programas de aplicación que pueden ejecutarse en más de una plataforma, por un lado con respecto a los procesadores para los sistemas operativos y por el otro a los sistemas operativos para los programas de aplicación. Esta portabilidad a través de plataformas múltiples es un objetivo importante para muchos desarrolladores de software libre (es decir, software que está disponible sin coste alguno y para el cual prácticamente no hay restricciones en cuanto a su uso).

Ante tanta diversidad de plataformas, podemos encontrarnos con que una es mejor que otra dependiendo de cuál sea nuestro objetivo, ya sea el desarrollo de aplicaciones (lo más usual), crear servidores web, instalar una base de datos, ejecutar un potente programa matemático o una herramienta gráfica. Gracias a la computación Cloud existe total libertad de elección de plataforma en función de nuestras necesidades y nos exime de estar obligados a integrar toda la base de trabajo en una única plataforma. Podemos optar por Linux RedHat, Ubuntu, Windows, Solaris, Mac OSX o un sistema IBM, una arquitectura x86, de 64 bits, un *mainframe* o un supercomputador independientemente de la máquina remota desde la que ejecutemos nuestra plataforma. Sin duda, esta es una de las grandes ventajas de la versatilidad de la computación Cloud.

### 1.4.3 API

Una interfaz de programación de aplicaciones (API) es un conjunto de normas y especificaciones que sigue un programa de software para acceder y hacer uso de los servicios y recursos proporcionados por otro programa de software especial que implementa la API (en este caso, el entorno *software* de una red Cloud). Sirve como una interfaz entre los programas de software diferentes y facilita su interacción, de forma similar a la interfaz de usuario facilita la interacción entre humanos y computadoras.

Las APIs Cloud especifican la manera en la que las aplicaciones y su código fuente interactúan con la Nube. Son un mecanismo mediante el cual el software puede solicitar información de una o varias plataformas Cloud a través de una interfaz directa o indirecta. En su gran mayoría, suelen estar escritas mediante los estilos de arquitectura de software REST y/o SOAP [JAKL]. El primero es el empleado por sistemas de hipermedia (hipertexto más archivos) distribuidos como la "World Wide Web" y el

[9] http://technet.microsoft.com/en-us/library/bb742607.aspx#XSLTsection123121120120
[10] http://www.linfo.org/platform.html

segundo, SOAP ("Simple Object Access Protocol"), es una especificación de protocolo para el intercambio de información estructurada en la implementación de Servicios Web en redes informáticas.

Dado que la mayoría de los Cloud públicos se sustentan en la Web y en aplicaciones específicas para la misma, desde la que ofrecen utilidades de interfaz y gestión, gran parte de ellos usan estilo de arquitectura REST para sus APIs, como las APIs desarrolladas por los mismos proveedores Cloud (Amazon Web Services, Google App Engine, Microsoft Azure) o de creadores de plataformas como Red Hat Deltacloud, las cuales ofrecen un mayor nivel de abstracción. Las APIs se diferencian según el servicio Cloud (infraestructura, plataforma, aplicación), y todas ellas pueden combinarse.

### 1.4.4 Aplicaciones y servicios

El software es la capa más alta de este servicio y la que es realmente funcional y útil para el usuario o desarrollador. Podemos lanzar todo tipo de *software* como herramientas de desarrollo de aplicaciones, servidores, bases de datos, sitios web, programas de cálculo, aplicaciones sobre navegadores como correo electrónico, procesadores de texto, grupos de trabajo, *"streaming"* multimedia, redes sociales y un sinfín de posibilidades al alcance de cualquier usuario de Internet.

## 1.5 Tipos de Cloud

Existen diversos tipos de nubes atendiendo a las necesidades de las empresas, al modelo de servicio ofrecido y a como se desplieguen en las mismas.

Dependiendo de donde se encuentren instaladas las aplicaciones y qué clientes pueden usarlas tendremos nubes públicas, privadas o híbridas, cada una de ellas con sus ventajas e inconvenientes [ALCO].

### 1.5.1 Nube pública

Esta denominación describe la Computación Cloud en el significado más aceptado de la palabra. Los servicios se encuentran en servidores externos al cliente, sea un particular, empresa o grupo de industria pudiendo tener acceso a los datos y aplicaciones vía aplicaciones web de forma gratuita o pagando una determinada cuota.

Estos servidores externos son mantenidos por terceros que venden servicios en la Nube (empresas de "*Cloud Hosting*") y el cliente desconoce la localización exacta de los que él mismo ha contratado o si los equipos donde se almacenan sus datos son compartidos por otros clientes.

**Ventajas**: Se obtiene gran capacidad de almacenamiento y procesamiento sin instalar máquinas físicamente, ahorrando en gasto inicial de inversión y en mantenimiento y seguridad, los cuales recaen bajo la responsabilidad del proveedor, haciendo que el riesgo por adoptar esta tecnología sea bajo y que el retorno de la inversión sea más rápido. Es muy conveniente adoptar este tipo de nube cuando se requiere que una cantidad de usuarios considerable genere carga de trabajo de una aplicación web, se necesite desarrollar y depurar código de proyectos colaborativos o se necesita una capacidad de almacenamiento incremental, entre otros ejemplos.

**Inconvenientes**: Aparte de la cierta inseguridad que pueda producir el hecho de dejar cierta información en manos de terceros, están la dependencia a los servicios en línea (vía Internet), la dificultad que la integración de estos servicios con otros sistemas propietarios mediante API's pueda generar en función de su madurez funcional y la velocidad de acceso si se emplean protocolos seguros como HTTPS.

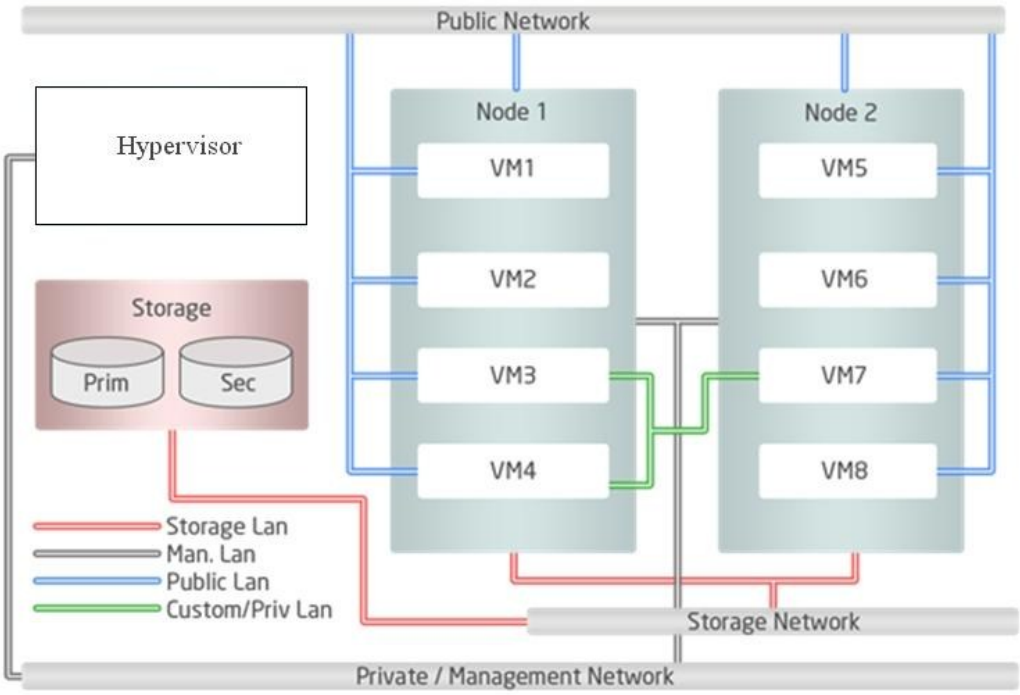


Figura 1.13: Nube pública. Fragmento de la infraestructura necesaria para formar una nube pública.

En la Figura 1.13 se pueden observar dos nodos (máquinas físicas) con cuatro máquinas virtuales instanciadas en cada uno. Existe una red para el control y monitorización de las máquinas virtuales que conectan la máquina física con SO hipervisor a los nodos, otra para el tráfico de datos almacenados desde el hipervisor hacia los nodos (ambos interconectados), y otra dirigida a la conexión con la red pública. En verde observamos la simulación de una conexión virtual privada entre tres máquinas (3, 4 y 7) de un mismo cliente.

### 1.5.2 Nube privada

La infraestructura es operada para una sola organización. Puede ser administrada por la organización o por un tercero y puede existir dentro de la misma ("*on premises*") o fuera de la misma ("*off premises*"). En general, una nube privada es una plataforma para la obtención de hardware, almacenamiento e infraestructura de red (IaaS), pero también se puede tener una nube privada que permita desplegar aplicaciones (PaaS) e incluso aplicaciones en sí mismas (SaaS).

Como apunte, señalar que los conceptos 'nube' y 'centro de datos privado' fueron acuñados hace ya 50 años en publicaciones como la anteriormente mencionada "*The Challenge of the Computer Utility*" (Douglas Parkly, 1966) y que por tanto el término de 'nube privada' es considerado por muchos como un mero neologismo.

**Ventajas**: Localización de los datos dentro de la empresa, lo cual otorga mayor seguridad de los datos y hace más fácil la integración de estos servicios con otros sistemas propietarios. Basadas en la virtualización, estas nubes proveen capacidad de recuperación frente a fallos y de escalabilidad personalizada. Son muy adecuadas si el negocio de la empresa se basa en aplicaciones y datos que requieren estricta seguridad (empresas de TI) y son suficientemente solventes para desarrollarlas.

**Inconvenientes**: La inversión inicial en infraestructura física, sistemas de virtualización, ancho de banda y seguridad conlleva una pérdida en escalabilidad, y supone un retorno más lento de la inversión. A eso hay que añadir los gastos de mantenimiento.

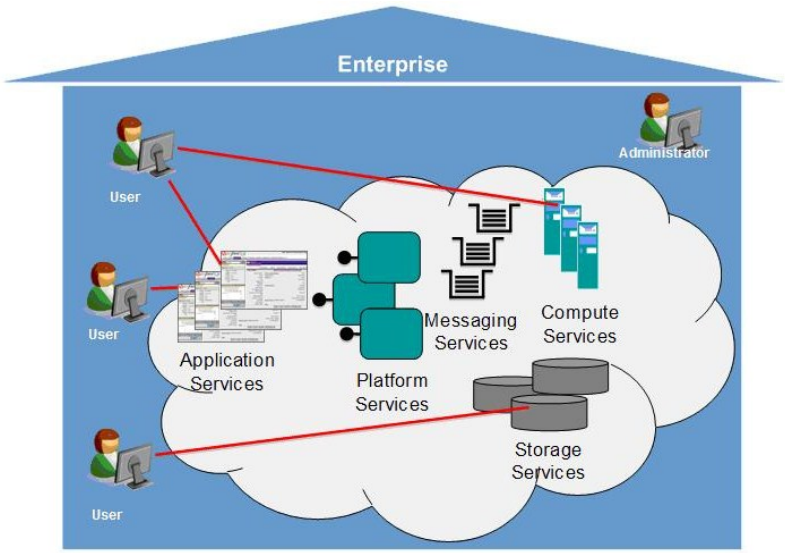


Figura 1.14: Nube privada "*on premises*". Las utilidades se proporcionan a través de una intranet por detrás del firewall de la empresa propietaria, que administra la nube. El acceso está limitado a los clientes y su red de socios, como máximo. De esta manera se aumenta la eficiencia manteniendo una mayor personalización y control dentro de la organización. En un ambiente de nube privada, todos los recursos son locales y dedicados.

### 1.5.3 Nube comunitaria

Estas nubes son una variante de las nubes públicas. La infraestructura de esta nube es compartida por varias organizaciones públicas o privadas y apoya los intereses propios de una comunidad particular sobre un campo específico (seguridad, investigación...). Por tanto, comparten las mismas características ya descritas con las nubes públicas con la siguiente excepción: apoya las preocupaciones de una comunidad en particular con unos costes a medio camino entre los generados por contratar a empresas dedicadas a servicios Cloud o contar con una infraestructura privada, se cuenta con ciertos niveles de privacidad, seguridad y/o política de cumplimiento, como la "*Gov Cloud*" solicitada por el gobierno de EE.UU. a Google en 2009.

### 1.5.4 Nube híbrida

Es la composición de dos o más nubes, por ejemplo privada y pública, que permanecen como entidades únicas pero que coexisten por emplear tecnologías que permiten compartir datos o aplicaciones entre las mismas, o dicho de otra forma, la combinación de servidores-instancia de nubes virtualizadas y hardware físico propio. El término inglés más correcto para expresa unión de dos o más nubes es "*combined cloud*".

Muchas empresas han concluido en que es más económico almacenar datos y *backups* en nubes públicas como Amazon S3 y que este modelo se presta también a un enfoque incremental. También una nube híbrida es un buen método de pasar la mayor parte de los recursos a la nube pública, ya que conlleva un riesgo menor: se traspasan las aplicaciones más apropiadas (menos "confidenciales") y más tarde se mueven las que sean necesarias. En el momento necesario, utilizando las API's de las distintas plataformas públicas existentes, se tiene la posibilidad de escalar la plataforma todo lo que se quiera sin invertir en infraestructura para aquellas necesidades que, por ejemplo, sean puntuales o intermitentes, no requieran sincronización o bases de datos complejas.

Esto permite a una empresa mantener el control de sus principales aplicaciones, al tiempo de aprovechar el Cloud Computing en los lugares donde tenga sentido a la vez que se obtienen las ventajas de las nubes públicas y privadas si un coste inicial elevado. Es por ello que esta práctica se está extendiendo considerablemente.

Un ejemplo práctico podría ser un escenario en el cual una aplicación se desarrolla y se prueba en una nube privada y luego se despliega a una nube pública. Otro es el de una compañía que desea usar una aplicación SaaS como estándar con seguridad: lo ideal será crear una nube privada con *"firewall"* y una VPN (red privada virtual) para mayor seguridad. Por último, también es muy indicada para modelos de negocio en los que se requieren un entorno online para que los clientes encarguen pedidos y comprueben su estado (nube pública) y una BB.DD. que gestiona la información de dichos clientes (nube privada).

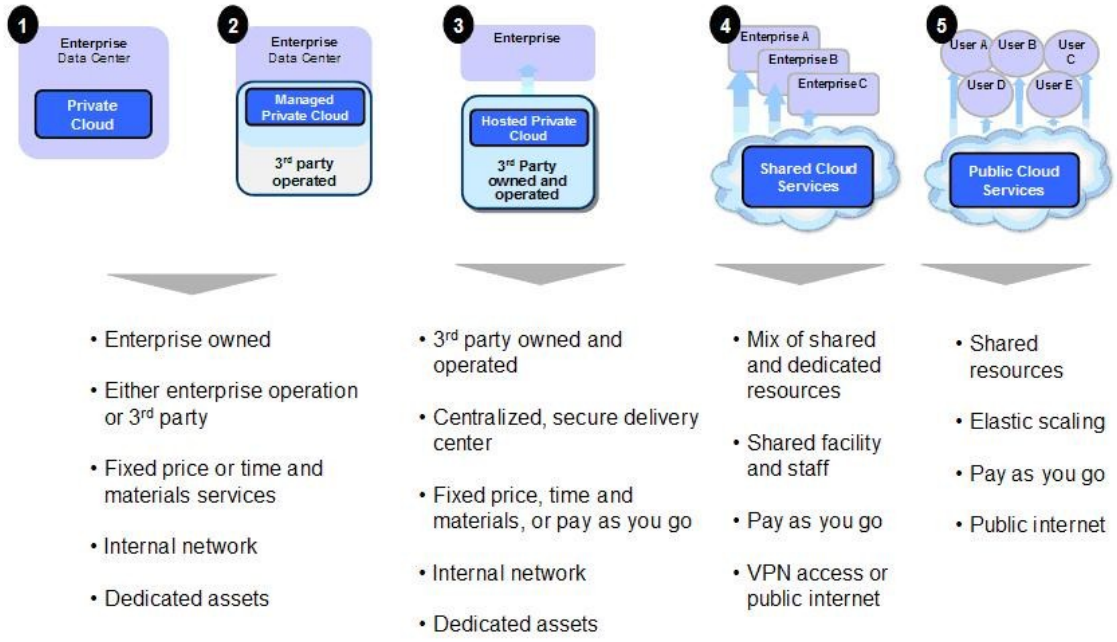


Figura 1.15: Modelos de suministro de servicio. Existen cinco modelos de Cloud según el tipo de nube. 1 – Nube privada exclusiva de una empresa auto-gestionada por ella misma. 2 – Nube privada exclusiva de una empresa gestionada por un tercero. 3 – Nube privada propiedad de un tercero que provee de servicios a una empresa. 4 – Nube comunitaria compartida con un número determinado de empresas. 5 – Un proveedor público de servicios Cloud ofrece recursos dinámicamente escalables. Así mismo, toda empresa puede formar parte o no de nubes públicas o privadas (nube híbrida)[11].

# 1.6 Servicios Cloud

Las redes de computación Cloud, ya sean públicas, privadas o híbridas cuentan con diversos modelos de servicio. Las funciones de Internet a través de una serie de protocolos de red forman una pila de capas, como se muestra en la figura siguiente (o como se describe más detalladamente en el modelo OSI). Una vez que una conexión a Internet se establece entre varios ordenadores, es posible compartir servicios dentro de cualquiera de las siguientes capas.

[11] https://www.ibm.com/developerworks/mydeveloperworks/blogs/c2028fdc-41fe-4493-8257-33a59069fa04/entry/september_19_2010_1_45_pm7?lang=en

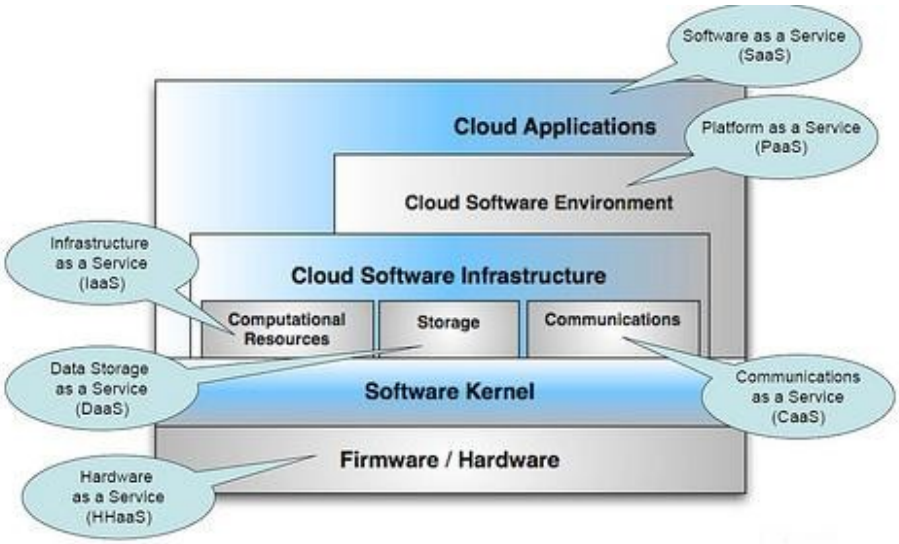


Figura 1.16: Capas de la computación Cloud. El nivel de abstracción de los diferentes modelos de servicio difieren según las capas OSI empleadas para la conexión entre máquinas y por tanto en los datos enviados en dicha conexión. De este modo se pueden suministrar por la Red servicios a bajo nivel simulando de este modo una máquina física *in situ* o a niveles superiores hasta la manipulación directa de aplicaciones sin importar cómo el proveedor ha implementado las capas inferiores [URQU].

El término "servicio" define una tarea que ha sido empaquetada de manera que pueda ser automatizada y sea entregada a los clientes de una manera reiterada y consistente. Estos servicios pueden ser proporcionados por un proveedor de servicios Cloud o a través de su propio centro de datos interno.

En la realidad, hay una mezcla entre los tipos de modelos de prestación de servicios que ofrecen los proveedores de redes Cloud. Por ejemplo, un proveedor de software como servicio (SaaS) podría optar por ofrecer distintos servicios de infraestructura (IaaS) a los clientes. El propósito de agrupar estos servicios en diferentes modelos es para ayudar a comprender lo que está por debajo de los distintos servicios disponibles en una nube. Todos estos modelos requieren la prestación de servicios de gestión y administración (incluyendo la seguridad) [HURW].

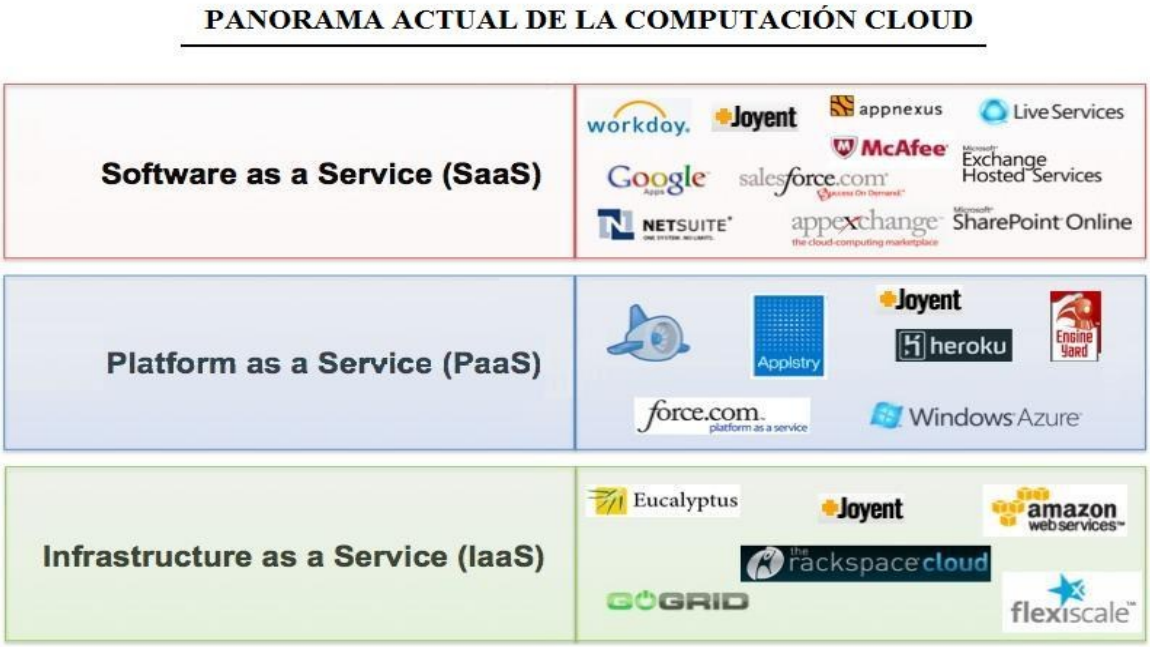


Figura 1.17: Panorama actual. Muchos proveedores de servicios en la Nube diversifican su modelo de negocio a todos los niveles. De esta manera recuperan el capital invertido en su infraestructura y obtienen beneficios de una forma más rápida al estar presentes en más de un mercado. Como podemos observar, este es el caso de compañías como 36orcé.com, Microsoft, Google y Joyent.

### 1.6.1 Infraestructura como servicio (IaaS)

Nos encontramos en obligación de hacer una mención especial a la infraestructura como servicio (IaaS) por el hecho de que, además de ser uno de los servicios Cloud más extendidos –si no es el que más-, es el empleado en nuestro programa de factorización de claves RSA para distribuir la carga de trabajo en equipos más potentes que un ordenador personal de prestaciones medias a través de Amazon EC2.

La infraestructura como servicio puede definirse resumidamente como la distribución de *hardware* (servidores, tecnología de redes, almacenamiento, y espacio físico para construir un centro de datos) como un servicio. También suele incluir varias opciones de elección para la gestión de sistemas operativos y tecnologías de virtualización para gestionar estos recursos o bien software arbitrario a la elección del consumidor, pero básicamente IaaS ofrece la posibilidad de desarrollar y ejecutar aplicaciones en una máquina con *hardware* muy potente bajo demanda desde un terminal que cuenta con recursos más limitados, lo que lo hace ideal para emplear herramientas de desarrollo o de cálculo que consumen un alto grado de procesamiento, memoria y almacenamiento.

Los clientes, en vez de invertir gasto en máquinas físicas, alquilan servicios de computación y pagan en función de los recursos consumidos. Es interesante destacar que gracias a esta posibilidad, la computación a cualquier nivel se democratiza: desde un particular interesado en usar brevemente esta tecnología, hasta grupos de

investigación con recursos económicos moderados y por supuesto empresas medianas y grandes corporaciones (vinculadas o no a las TI) tienen a su disposición este servicio de manera personalizada y a un precio muy asequible evitando grandes inversiones de infraestructura a corto plazo [SUNG].

Para obtener una configuración "personalizada" de IaaS, se incluye el denominado "escalaje dinámico", en el cual se aumentan o disminuyen los recursos alquilados de forma automática. Por poner un ejemplo, el contrato de un servicio IaaS puede especificar que los recursos estarán disponibles el 99.999% del tiempo efectivo contratado y que los recursos provistos se incrementarán dinámicamente si más del 80% de un recurso en concreto está siendo utilizado.

IaaS encaja perfectamente para las empresas con proyectos de investigación intensiva o I+D. Los servicios Cloud permiten a los equipos de investigación científica (matemáticas, física, medicina, astronomía, etc.) desarrollar pruebas y análisis a niveles que de otra manera serían imposibles de realizar sin acceder a una infraestructura de centro de datos, supercomputadores o "*mainframes"*.

Éstas y otras organizaciones con necesidades similares de recursos de computación podrían optar por adoptar como un servicio el *hardware* necesario para sus propósitos: una red servidores, tecnología de conexión de red, almacenamiento y el espacio necesario para estas instalaciones. En lugar de invertir capital en una infraestructura que cubra sus mayores niveles de demanda, comprarían la capacidad de computación deseada en función de sus necesidades circunstanciales.

Algunos servicios IaaS son Flexiscale, One Source, Rackspace, Sungard, CloudScaling y los ofrecidos por las grandes compañías de TI como IBM, Vmware y HP, por citar algunas. Por encima de ellas destaca Amazon's Elastic Cloud por ser la pionera dentro de este campo.

#### 1.6.1.1 Amazon EC2

Dentro del panorama actual, el ejemplo más destacado dados su rendimiento en relación a su coste y éxito comercial es Amazon's Elastic Cloud (Amazon EC2). Mediante una interfaz a través de un portal web, los usuarios acceden a máquinas virtuales y, mediante la escalabilidad, EC2 ofrece recursos específicos con tarifas por horas. El servicio es denominado "elástico" al contar con la opción de modificar al instante la infraestructura requerida: el usuario debe hacer una petición para ello, por ello el servicio no se considera dinámicamente escalable. Eso sí, puede elegir el SO deseado (Linux, Solaris y Windows).

EC2 emplea el monitor de máquina virtual Xen para crear y gestionar sus máquinas. Xen es una capa de software abierto relativamente sencilla desarrollada por la Universidad de Cambridge que permite a otros SO ejecutarse en el mismo sistema. De este modo, el servicio de Amazon incluye la creación de servidores virtuales en función de varios tipos de especificaciones técnicas.

(Nota: En el siguiente capítulo se dará una descripción más detallada de este servicio de Computación Cloud.)

## 1.6.2 Otros: PaaS, NaaS, SaaS y XaaS

A continuación se enumeran y describen otras de las principales formas de computación Cloud existentes de las cuales derivan la mayoría de servicios disponibles.

### 1.6.2.1 PaaS (Platform as a Service)

Cuando un proveedor suministra una "plataforma como servicio" a un cliente, está ofreciendo más que simple infraestructura Cloud. A la capacidad computacional y de almacenamiento, hay que añadir un conjunto integrado de software que incluye todo lo que necesita un desarrollador para construir una aplicación. Dicho de otra manera, PaaS es la encapsulación de una abstracción de un ambiente de desarrollo y el empaquetamiento de una carga de servicios.

También puede ser interpretado como una evolución del "*web hosting*". Durante los últimos años, las compañías de "*web hosting*" han provisto al contratante de un conjunto de *software* para desarrollar sitios web. La plataforma como servicio va un paso más allá al incluir formas de gestión del desarrollo de la aplicación en todas sus fases: desde la planificación y diseño, construcción y despliegue, testeo y mantenimiento. La principal característica de PaaS consiste en contar con software de desarrollo y despliegue basados completamente en el entorno Cloud. Esta capacidad le permite al consumidor desplegar en la infraestructura del proveedor aplicaciones creadas por el primero, incluso adquiridas, usando lenguajes de programación y herramientas del proveedor. Por tanto, no se requieren inversiones de gestión y mantenimiento extra por parte del cliente, todo ello viene incluido en el servicio. Todos las fases concernientes al desarrollo de una aplicación, desde el diseño al testeo, viven en la nube.

PaaS tiene la capacidad inherente de ser compatible con cualquier estándar dentro del servicio web y suele ser suministrado al cliente con escalabilidad dinámica. Dentro del PaaS, esta cualidad se refiere en concreto a que el software es dinámicamente escalable: lo cual quiere decir que se pueden ajustar a medida el acceso y la seguridad de la información.

Aunque este servicio otorga muchas facilidades a sus usuarios, cuenta con algunos inconvenientes. El mayor problema de la plataforma como servicio es su portabilidad a otra forma de PaaS, es decir, la posibilidad de encasillarse en el empleo de un entorno de desarrollo en particular y sus componentes. Los servicios PaaS cuentan con elementos propietario únicamente compatibles con ese servicio en concreto como herramientas de desarrollo o librerías. En consecuencia, contratar un servicio PaaS puede desembocar en un fuerte compromiso con el propietario de la plataforma y en la incapacidad de migrar las aplicaciones a cualquier otra sin modificarlo o reescribirlo en un cierto grado. Es decir, si un cliente está insatisfecho con el servicio contratado, éste posiblemente deberá afrontar grandes gastos para realizar los cambios convenientes para que su *software* pueda satisfacer los requisitos de otro proveedor de PaaS.

El temor a que esto pueda ocurrir ha desencadenado un auge de una nueva forma de PaaS emergente: la plataforma de servicio de código abierto. Ofrece los mismos servicios que cualquier PaaS, con la ventaja de que evade la posibilidad de fijarse a una

plataforma. Ejemplos de este tipo de PaaS son Google App Engine, AppJet, Etelos, Qrimp, Force.com, Windows Azure o la española EyeOS.

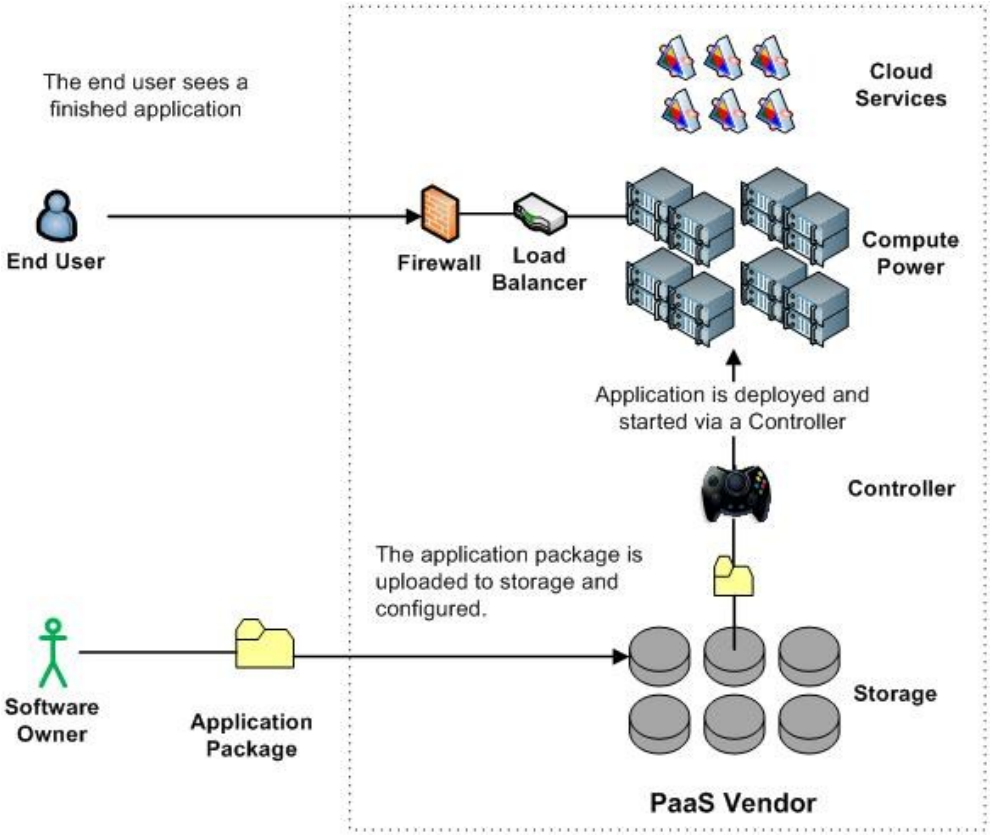


Figura 1.18: Plataforma como servicio. PaaS incluye todas las facilidades al programador para prototipar, analizar, desarrollar, testear, documentar y poner en marcha aplicaciones en un sólo proceso. Además da servicio de integración de la base de datos, seguridad, escalabilidad, almacenaje, copias de seguridad, versioning, y facilidad para colaborar en la comunidad.

Actualmente, los proveedores de PaaS comercializan entornos especialmente diseñados para el desarrollo de *software* o bien para la ejecución de programas altamente sofisticados, con fines empresariales, científicos, de investigación o académico. Sin embargo, se están desarrollando ciertas plataformas que supondrán el inicio de la expansión de PaaS a los hogares, a través del ocio electrónico: televisión digital y videojuegos.

En EE. UU. Se están comercializando algunas plataformas que ofrecen miles de canales en alta definición como AOL TV y MSN TV. Además, muchas grandes compañías del sector de las telecomunicaciones están investigando para lanzar al mercado nuevos productos que podrían incluir la visión en 3D, algunos nuevos ejemplos de la denominada "Cloud TV" son Sony y Google [CLTV]. Otras plataformas como OnLive (funcionando ya en EE.UU.) y Gaikai, ofrecerán dentro de unos años centenares de videojuegos prescindiendo del formato físico por una cuota anual muy económica[12]. Todos estos productos proporcionan a su vez redes sociales de comunicación y diversas aplicaciones. Su mayor problema para consolidarse reside en la actual infraestructura de banda ancha a nivel mundial, aún por debajo de los

[12]http://www.onlive.com/; http://www.gaikai.com/

requisitos necesarios en general, por lo que se buscan formas de optimización de este servicio. Con estos ejemplos, encontramos de nuevo otras formas de Cloud de propósito general.


Figura 1.19: Ejemplo de diferentes servicios de plataforma. Google Apps Engine proporciona todo tipo de software de desarrollo y aplicaciones, mientras que Amazon EC2 es muy versátil a costa de que sea la propia comunidad la que proporcione soporte[13].

Dentro del ámbito europeo, se desarrollan también proyectos internacionales como 4CaaSt, una iniciativa financiada con 15millonesdeeuros por la UE (Acuerdo de Subvención de la UE nº. 258.862) basada en el software abierto de OpenNebula, implicada en el área de servicios en Internet y la virtualización. El software está destinado a crear una plataforma avanzada de Cloud que soporte el óptimo y elástico alojamiento de aplicaciones escalables sobre las diferentes capas de protocolo de la Red.[14]

#### 1.6.2.2 NaaS (Network as a Service)

El modelo Naas, el más reciente de la industria, se centra actualmente en los operadores de redes fijas y móviles. La "red como servicio" ofrece la posibilidad de que el consumidor, por lo general un usuario de la empresa o desarrollador de aplicaciones, utilice capacidades de red y datos. Por citar algunas; control de llamadas, mensajería y facturación, y la información del suscriptor; ubicación, presencia y perfil, todo a través

[13] http://www.zdnet.com/blog/hinchcliffe/comparing-amazons-and-googles-platform-as-a-service-paas-offerings/166?tag=btxcsim
[14] http://www.4caast.eu/

de APIs de programación Web 2.0 para mejorar sus aplicaciones. Los integrantes de este servicio son los componentes de la red y el Service Delivery Platform (SDP) o "plataforma de envío de servicio", que suministra la información requerida creando un entorno Cloud móvil. Además de las APIs, NaaS también ofrece la ejecución de servicios en la nube para la creación de "mash-ups" (sitios o aplicaciones web que usan contenido de otras web para crear un nuevo contenido, a través de HTTP y usando API's o sindicadores web como RSS). También puede servir para aumentar la potencia en el procesamiento de aplicaciones que un dispositivo móvil no podría ser capaz de manejar. Ejemplos de proveedores de Naas son: AT & T, Orange Partner y Vodafone Betavine [RAVI].

El concepto de "red como servicio" también apunta por otro cauce totalmente distinto encaminado a la "banda ancha por demanda": algunos proveedores de red comienzan a ofrecer un servicio elástico de conexión a Internet o telefonía. Consiste en *software* de algoritmos de control de tráfico de información para evitar la sobrecarga en ciertos puntos de la red y conseguir así un reparto mejor distribuido y uniforme de datos, posibilitando la posibilidad de aumentar o disminuir la capacidad de la conexión a la red (escalabilidad). Este planteamiento es llevado a la práctica en Cloud públicos y privados, a través de WAN. Los algoritmos de este tipo permiten la transmisión de datos sobre protocolo UDP, el cual no está orientado a conexión de la misma manera que TCP/IP (no garantiza la entrega), por lo que UDP es significativamente más rápido al no requerir ACK tras el envío de paquetes.

He aquí algunos ejemplos pioneros de esta tecnología. El algoritmo de UCSD de la Universidad de California permite coordinar conjuntamente los limitadores de tasa de velocidad distribuidos para hacer cumplir los límites de la tasa global de ancho de banda de los usuarios y modifica dinámicamente la asignación de ancho de banda a través de múltiples sitios o redes, de acuerdo con la demanda actual de la red. HyperIP de NetEx, permite a las organizaciones con redes de datos compuestas por dispositivos Cisco ejecutar software para optimizar las redes WAN. Es compatible con los supervisores Vmware, HyperV y Xen [FISH]. Internet2, un consorcio sin ánimo de lucro de redes avanzadas en los Estados Unidos, está diseñando actualmente una nueva red (Dynamic Circuit Network o DCN) que utiliza tecnologías desarrolladas por la comunidad y basadas en protocolos estándar para proporcionar la opción de elegir rutas de conexión óptica entre dos extremos de la red. Actualmente se prueba en proyectos de investigación científica. El algoritmo permite la coordinación de tráfico de servicios basados en la Nube, y por tanto de su coste asociado. El diseño es escalable a cientos de nodos con una sobrecarga muy baja, y resiste tanto a delays como a la pérdida de comunicación[15].

Los proveedores de recursos basados en la Nube podrían controlar el uso de ancho de banda de red y su coste, como si todo el ancho de banda procediese de un solo canal. Para las redes de distribución de contenido que actualmente proporcionan servicios de replicación de sitios web a terceros, podría ser una poderosa herramienta para administrar el acceso a los contenidos del cliente. Bancos de pruebas como PlanetLab son a menudo invadidos por la demanda de usuarios de la red y esta tecnología podría hacer de éstos herramientas de investigación más eficaces[16].

[15] http://www.internet2.edu/network/dc/
[16] http://www.planet-lab.org/

Entre las aplicaciones actuales de la red de circuito dinámico, Internet2 espera facilitar la transferencia de datos del gran colisionador de hadrones del CERN a investigadores de otras instituciones, y ha hecho ensayos en los que ha conectado el acelerador con la Universidad de Nebraska [INTE]. En el futuro, se espera que se desarrollen a partir de la tecnología aplicaciones comerciales, como miles de canales de alta definición y vídeo bajo demanda. La comunidad científica y a posteriori, la sociedad, están a un paso de conocer una nueva era en el sector de las comunicaciones.

#### 1.6.2.3 SaaS (Software as a Service)

Uno de los primeros servicios Cloud implementados fue el SaaS. El "*software* como servicio" ofrece al cliente la posibilidad de utilizar las aplicaciones del proveedor en una infraestructura de nube. Algunos ejemplos ampliamente extendidos a nivel usuario hoy en día que pueden ser usados de forma gratuita son el correo electrónico, portales y mensajería instantánea, aunque obviamente también se ofrece *software* específico a empresas. El consumidor no administra ni controla la infraestructura que soporta estos servicios, pero si algunos parámetros de configuración.

SaaS tiene sus orígenes en los primeros servicios de "*hosting*" llevados a cabo por los Proveedores de Servicios de Aplicación (ASPs). El mercado de los ASPs creció muy pronto con la explosión de Internet con el desarrollo por parte de las grandes compañías de aplicaciones de correo y mensajería privadas y seguras para satisfacer las necesidades de comunicación dentro de la propia empresa. Gracias a su facilidad de uso pasaron fácilmente a ofrecerse como un servicio muy útil y económico. Existen dos formas de SaaS:

- **Multipropiedad simple**: cada cliente cuenta con sus propios recursos suministrados por el ASP, los cuales se encuentran separados de aquellos de los demás clientes. Es una forma de multipropiedad relativamente ineficiente.

- **Multipropiedad de grano fino**: Este modo ofrece el mismo nivel de separación de recursos, pero desde el punto de vista de la ingeniería del *software*, es bastante más eficiente. Todos los recursos son compartidos, pero la configuración de acceso y el almacenamiento de datos del cliente son independientes de la aplicación, lo cual permite una escalabilidad más versátil.

Cuando los usuarios hacen exactamente las mismas operaciones dada una aplicación SaaS, el coste por usuario es tan bajo que el servicio puede ser ofrecido gratuitamente. A esto se le llama SaaS de "escalabilidad masiva" Un ejemplo es Yahoo! Mail, con más de 260 millones de usuarios. Otros entornos diseñados para esta forma de SaaS son Facebook, eBay, Skype, Zoho y Google Apps.

A la hora de ofrecer aplicaciones a nivel empresarial, ya sean referentes a contabilidad, colaborativas, gestión de proyectos, pruebas, análisis, marketing online, seguridad y herramientas de comunicación con el cliente (CRMs), IBM, HP, Microsoft y Vmware[17] son los principales de este mercado, siendo Salesforce.com el máximo exponente.

[17] http://www-01.ibm.com/software/es/lotus/saas/
https://portal.saas.hp.com/site/Portal.do
http://www.microsoft.com/serviceproviders/saas/default.mspx,
http://www.vmware.com/company/news/releases/saas-tradebeam.html

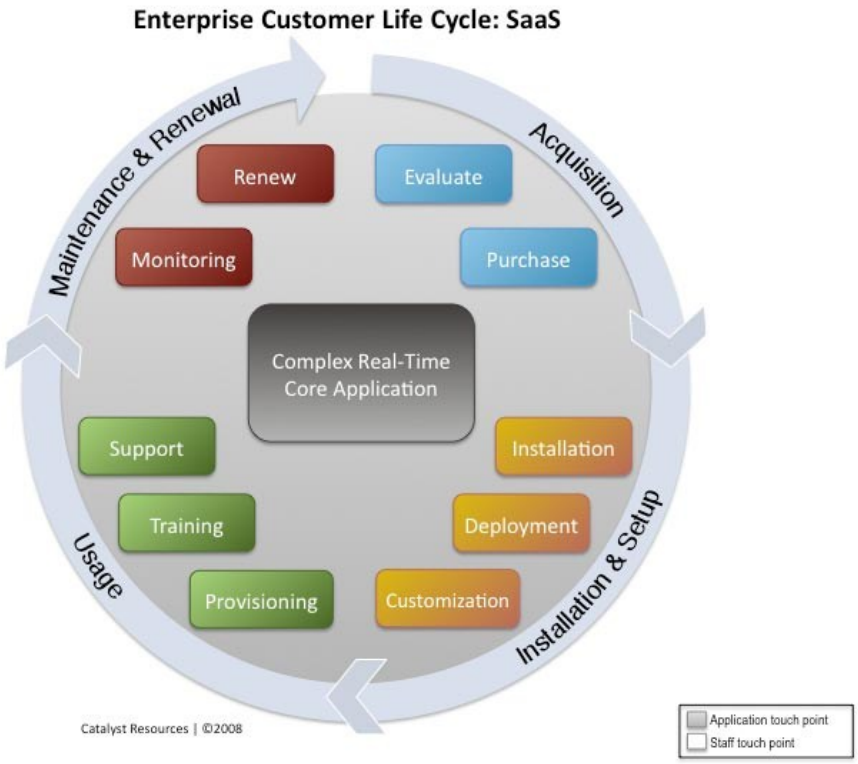


Figura 1.20: Ciclo de vida de SaaS.

Para una empresa que decide contratar los servicios de un proveedor SaaS, la correcta elección es un factor clave dado que el funcionamiento de una aplicación técnica en la Nube requiere un largo proceso. Por esto mismo, en primer lugar la empresa suele disponer de un periodo de evaluación. Ya conforme, compra la aplicación, la instala, despliega y configura según los requisitos que su negocio exija. Después los usuarios deben aprender su funcionamiento con ayuda de soporte técnico. Finalmente, la empresa debe evaluar su rendimiento continuamente y actualizarlo o renovarlo si es preciso.

#### 1.6.2.4 XaaS (X (Anything) as a Service)

XaaS es un término colectivo empleado para referirse a las expresiones anglosajonas "*X as a service*", "*anything as a service*" o "e*verything as a service*"- "X como servicio", "cualquier cosa como servicio" o "todo como servicio". Este acrónimo ha surgido como producto del creciente número de servicios que son suministrados a través de Internet en lugar de hacerlo localmente mediante programas o herramientas físicamente. XaaS es el origen y la esencia de la computación Cloud y el concepto de ser capaz de invocar componentes reutilizables y de grano fino de software a través de una red junto con la interacción orquestada entre los diversos componentes para suministrar un servicio: desde una funcionalidad de comunicación social, aplicaciones de ofimática, ocio electrónico, un problema de negocio o servicio a empresas. El proveedor será el punto de entrada de las soluciones ofrecidas a través de la Nube.

Actualmente muchas compañías investigan sobre este campo. Con su experiencia en hardware e Internet, gigantes como Google, Microsoft y Salesforce pueden convertirse en los grandes proveedores de este servicio (HP es el mayor promotor del modelo de

EaaS)[18]. Este modelo tiene como propósito principal superar los retos técnicos que surgen al migrar las operaciones de un negocio completamente a Internet, o lo que es lo mismo, a la Nube [SHRO]. Por tanto, aún no está clara la definición de este concepto ni nadie tiene claro cuáles son las posibilidades y los límites del "todo como servicio".

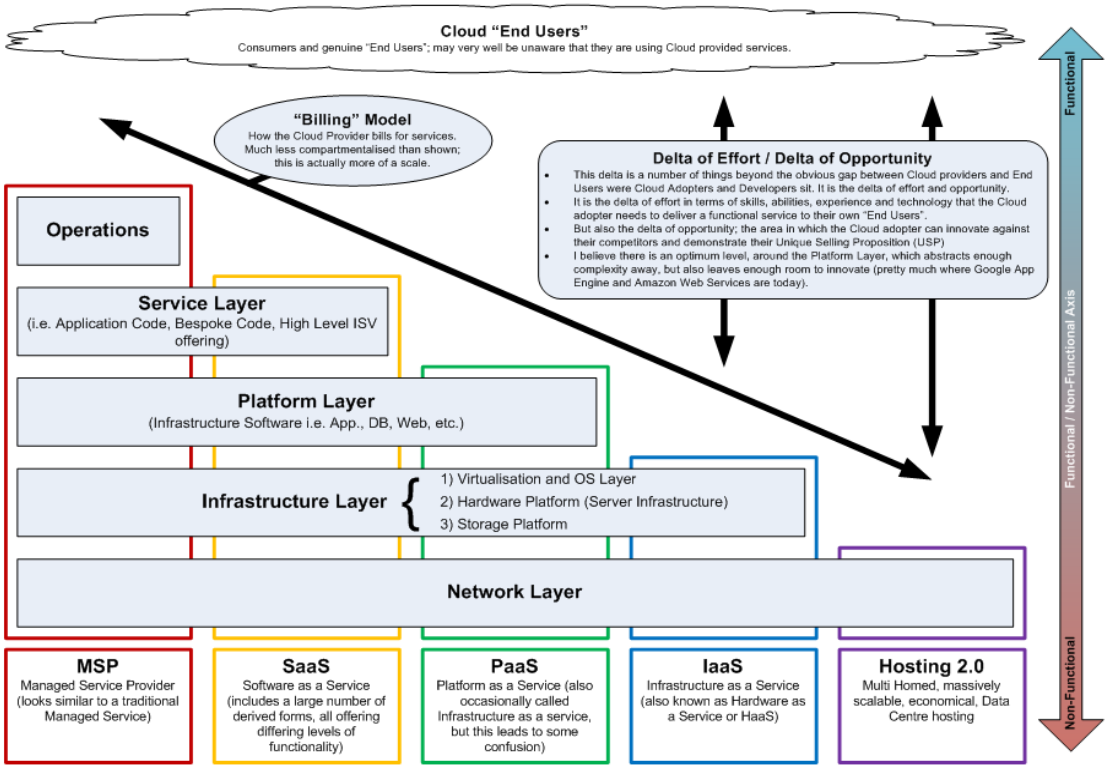


Figura 1.21: Todo puede ser computación Cloud. Dependiendo de dónde esté definida la frontera que divide la parte de la red que engloba a la Nube gestionada por el proveedor de servicios y la infraestructura privada invertida por el usuario, se define un tipo de servicio. Dicho de otra manera, tanto si disponemos de un centro de datos privado con tecnología de virtualización conectado a la red como si usamos un ordenador personal, teléfono móvil o televisor con conexión a Internet, se puede decir que estamos inmersos en la Nube (sin ser conscientes exactamente de ello en la mayoría de las ocasiones). Esta perspectiva engloba de forma generalizada el concepto de XaaS.

Sin lugar a dudas, esto marca una nueva era en la computación. "Todo como servicio" (XaaS o EaaS) es considerado por muchos como la próxima palabra de moda o "*buzzword*". El futuro de la computación está en la Nube de Internet. La industria de la tecnología está cambiando a un nuevo modelo en el que las personas y las empresas ya no necesitarán instalar paquetes de aplicaciones de software en sus equipos. En su lugar, utilizarán su navegador web para acceder a una amplia gama de "servicios en la Nube", disponibles bajo demanda a través de Internet. Desde un punto de vista más escéptico, la computación en nube ha sido criticada por limitar la libertad de los usuarios y hacerlos dependientes del proveedor de servicios. Algunos críticos afirman que sólo es posible usar las aplicaciones y servicios que el proveedor esté dispuesto a ofrecer. El tiempo dirá si el modelo EaaS se impondrá, si no cumplirá con las expectativas o si el PC y la red Cloud convivirán simultáneamente.

[18]Hewlett-Packard Development Company: http://www.hp.com/hpinfo/initiatives/eaas/index.html

Ahora bien, como todo nuevo avance científico o tecnológico, corresponde a la Humanidad darle un uso adecuado, útil y próspero que mejore la calidad de vida de todos sus integrantes, respetando los derechos y deberes de cada uno de ellos. La "tecnología informática como un servicio" puede significar la integración de todas las personas en una red global interconectada a un nivel muy superior al ofrecido hasta ahora por Internet: su evolución dependerá de las decisiones tomadas por los organismos gubernamentales, las grandes compañías tecnológicas, la comunidad científica y el conjunto de los ciudadanos.

# 2. Amazon

## 2.1 Historia de Amazon

Amazon es una compañía estadounidense con sede en Seattle, Estado de Washington, líder global en el comercio electrónico, que ofrece todo tipo de servicios informáticos a sus clientes. Una de las primeras empresas en vender a través de internet, siendo objetivo de ésta ser la mejor tienda online, dándole al comprador una buena experiencia.

Fue Jeff Bezos [BEZO], quien después de abandonar su trabajo como programador en Wall Street, se fijó como objetivo construir la mejor tienda online del mundo, no la primera, la mejor. Con la idea obsesiva de considerar al comprador como la mejor experiencia comercial**.** En 1994 fundó, en un garaje de Seattle, la empresa Cadabra.com. El 16 de julio de 1995, Amazon.com abre sus puertas al público, una librería online, con un promedio de 200,000 títulos, los cuales estaban disponibles no solo por internet, sino también por e-mail [BOGG].

Como parte de los cambios de Cadabra.com, Bezos considero importante darla a conocer con un nombre nuevo, fácil de identificar y así nació Amazon, cuyo nombre deriva del rio Amazonas. La razón por la elección de Amazon como nombre fue debido a que Amazon es el río más largo del mundo, y Amazon.com recuerda al propósito inicial que tenía Jeff Bezos para Amazon, quería dar a entender que Amazon sería la librería más grande del mundo. Por otro lado, se rumorea que el cambio de nombre de Cadabra.com por Amazon, se debió a que en ese momento los buscadores, en especial Yahoo!, listaban sus resultados en listas ordenadas alfabéticamente, de forma que Amazon aparecería en los primeros lugares. Otro de los rumores del cambio de nombre fue porque Cadabra sonaba muy parecido a cadáver.

El crecimiento de Amazon.com fue de manera vertiginosa, logrando dos años más tarde, el 15 de mayo de 1997 entrar a la Bolsa de Valores de New York, específicamente a la NASDAQ bajo el título de AMZN y a un precio de 18 dólares la acción. Además, la prestigiosa revista Time Magazine calificó a Bezos como la persona del año en 1999, por ser dueño de Amazon, que se había vuelto muy popular. El primer plan económico de Amazon era inusual. La compañía no cambió nada en 4 o 5 años. Tiempo después, pensando en retrospectiva, la estrategia funcionó bien. Más tarde, la compañía se decidió a crecer, y lo hizo bien. Amazon ha absorbido numerosas empresas, al igual que Google o Microsoft. Algunas de estas adquisiciones son: Audible (una empresa de audio-libros), BookSurge (dedicada a los libros de baja demanda), Mobipocket (la cual crea e-books y dispositivos para libros electrónicos) o Fabric.com (una empresa de costura).

Para entender un poco más la industria de Amazon es importante mencionar la ideología para que su negocio se encuentre orientado al mercado dentro de la red. Estar encaminado al cliente, porque considera sus necesidades, posibilidades, prioridades, intereses, gustos, y deseos personales. No perder de vista, ni un segundo y para nada, a

la competencia. Garantizar la rentabilidad, aunque como inversión al principio se trabaje a pérdidas.

Desde 1998 Amazon lanzó el reto de convertirse en la mayor tienda del mundo .Ha ampliado considerablemente su oferta de productos, lugares internacionales y la red mundial de centros de servicio al cliente. Opera en sitios como Reino Unido, Alemania, Francia, Japón, Canadá, Italia y China.

A día de hoy el crecimiento de Amazon es de tal magnitud, que no sólo vende todo tipo de productos, sino que ofrece todo tipo de servicios informáticos. Los Servicios Web de Amazon (AWS - Amazon Web Services) fueron lanzados a principios del 2006.

Amazon Web Services[19] ha proveído a las compañías con una plataforma de infraestructura de servicios web en la Nube. Con AWS se puede solicitar alto poder en cómputo, almacenamiento y otros productos teniendo acceso a una gama de servicios de infraestructura en Tecnologías de Información. Amazon tiene una larga historia respecto a la infraestructura descentralizada, esto le dio acceso al equipo de Amazon la oportunidad de ofrecer recursos sobre demanda tales como computación y almacenamiento e incrementando sobre todo la productividad y agilidad. Para el 2005, Amazon había gastado casi una década y cientos de millones de dólares construyendo y administrando la grande, confiable y eficiente infraestructura TI que puso en marcha la operación de las plataformas rentables en línea más grande del mundo.

Amazon Web Services es una recopilación de servicios de infraestructura. Todos los servicios AWS pueden utilizarse de forma independiente o implementarse de forma conjunta para crear una completa plataforma informática en la Nube. Algunos de los servicios más importantes son:

- **Amazon Elastic Compute Cloud (Amazon EC2):** Un servicio web que ofrece capacidad informática con tamaño modificable en la Nube. El cliente define el entorno Amazon EC2 virtual con el sistema operativo, los servicios, las bases de datos y la pila de plataforma de aplicaciones que necesite su aplicación alojada. Amazon EC2 ofrece una consola de gestión completa y una API para la gestión de sus recursos informáticos.

- **Amazon Simple Storage Service (Amazon S3):** Una sencilla interfaz de servicios web que puede utilizarse para almacenar y recuperar grandes cantidades de datos, en cualquier momento, desde cualquier parte de la web. Concede a desarrolladores y a la empresa acceso a la misma infraestructura de almacenamiento de datos económica, altamente escalable, fiable y rápida que utiliza Amazon para tener en funcionamiento su propia red internacional de sitios web.

- **Amazon CloudFront**: Un servicio web que proporciona entrega de contenido de alto rendimiento distribuida a nivel internacional. Con Amazon CloudFront se ofrece la distribuir fácilmente o transmitir contenido a sus usuarios con poca latencia, elevadas velocidades de transferencia de datos, ningún tipo de compromiso y perfecta integración con Amazon S3.

[19] http://aws.amazon.com/es/

- **Amazon Relational Database Service (Amazon RDS):** Un servicio web que proporciona capacidad rentable y de tamaño modificable para implementaciones de MySQL en la Nube, que además gestiona áridas tareas como la copia de seguridad, el escalado y la aplicación de revisiones.

- **Amazon SimpleDB**: Un servicio web que permite ejecutar consultas sobre datos estructurados en tiempo real. Una de las características de Amazon SimpleDB es su fácil utilización, y el ofrecimiento de la funcionalidad básica de una base de datos: búsquedas en tiempo real y consultas sencillas de datos estructurados pero sin la complejidad operativa.

- **Amazon Simple Notification Service (Amazon SNS):** Un servicio web que facilita las tareas de configuración, utilización y envío de notificaciones desde la Nube. Al utilizar Amazon SNS tanto los desarrolladores como las empresas podrán enviar notificaciones o mensajes a las aplicaciones o a las personas, utilizando un mecanismo de "inserción" y que estos mensajes se entreguen a través del protocolo que deseen (i.e. HTTP, correo electrónico, etc.).

- **Amazon Simple Queue Service (Amazon SQS)**: Un sistema de colas seguro y de alto rendimiento que permite distribuir de una forma fiable el trabajo entre procesos de la aplicación. Con Amazon SQS los desarrolladores y las empresas pueden simplemente mover datos entre componentes distribuidos de sus aplicaciones que realizan diferentes tareas, sin perder mensajes ni exigir que todos los componentes estén siempre disponibles.

- **Amazon Virtual Private Cloud (Amazon VPC):** Un puente transparente y seguro entre la infraestructura de TI existente de una empresa y la nube de AWS. Amazon VPC permite a las empresas conectar su infraestructura existente a un conjunto de recursos informáticos de AWS aislados a través de una conexión mediante Red privada virtual (VPN), así como ampliar sus capacidades de gestión existentes, como por ejemplo servicios de seguridad, firewalls y sistemas de detección de intrusiones para incluir sus recursos AWS.

El emblema de Amazon (imagen de la izquierda), también ha ido cambiando a lo largo del tiempo. El logo de Amazon de 1995 estaba formado por una A mayúscula sobre un fondo azul.


En la actualidad el logo de Amazon difiere mucho con el original.

amazon.com

Figura 2.1: Logo actual de Amazon


Figura 2.2: Variaciones del logotipo de Amazon a lo largo del tiempo.

# 2.2 Cloud Público de Amazon EC2

Amazon Elastic Compute Cloud[20]es un servicio web que proporciona capacidad informática con tamaño modificable en la Nube. El objetivo principal es proporcionar la cantidad adecuada de capacidad de CPU, independientemente del hardware subyacente, haciendo que la informática web resulte más sencilla a los desarrolladores. Se pueden usar las instancias de servidor EC2 en cualquier momento, por el tiempo que se necesite y para cualquier propósito legal.

Un ejemplo de enfoque típico de infraestructura Cloud es Amazon EC2, basada en una colección de servidores de propósito general conectados por red estándar de área local (LAN) mediante tecnologías de conmutación. En la capa superior, el modelo de infraestructura como servicio (IaaS) (ver sección 1.6.1), [ARMB] se construye donde los usuarios tienen acceso a los recursos del sistema, mediante redes virtuales y los anteriormente mencionados servidores.

Amazon EC2 cambia la economía de la informática al permitir pagar sólo por la capacidad que realmente se está utilizando. Además proporciona a los desarrolladores las herramientas necesarias para crear aplicaciones resistentes a errores y para aislarse de los casos de error más comunes. Amazon EC2 permite obtener y configurar capacidad a través de la sencilla interfaz de servicios web con una fricción mínima. Proporciona un control completo sobre sus recursos informáticos y permite ejecutarse en el entorno informático acreditado de Amazon. Amazon EC2 reduce el tiempo necesario para obtener e iniciar nuevas instancias de servidor a cuestión de minutos, lo que permite escalar con rapidez su capacidad, aumentándola o reduciéndola, cuando cambien los requisitos informáticos. Se pueden habilitar varias instancias de servidor simultáneamente porque todo esto está controlado por APIs de servicio web.

Amazon EC2 presenta un auténtico entorno informático virtual, que permite utilizar interfaces de servicio web para iniciar instancias con distintos sistemas operativos, cargarlas con su entorno de aplicaciones personalizadas, gestionar sus permisos de acceso a la red y ejecutar su imagen utilizando los sistemas que desee.

[20]http://aws.amazon.com/es/ec2/

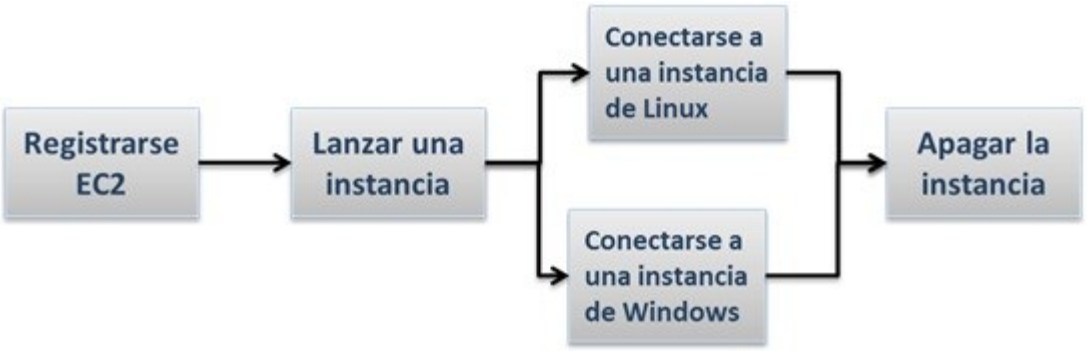


Figura 2.3: Esquema simplificado de uso de una instancia en Amazon.

Para utilizar Amazon EC2, es necesario:

- Seleccionar una imagen de plantilla pre-configurada para pasar a estar activo de inmediato. O bien crear una AMI (Amazon Machine Image) que contenga sus aplicaciones, bibliotecas, datos y valores de configuración asociados.
- Configurar la seguridad y el acceso a red en la instancia de Amazon EC2.
- Seleccionar los tipos de instancias y sistemas operativos que se desee y, a continuación, iniciar, finalizar y supervisar tantas instancias de su AMI como sea necesario, a través de las API de servicio web o la variedad de herramientas de gestión proporcionadas.
- Determinar si se desea una ejecución en varias localizaciones, utilizar puntos finales de IP estáticos o adjuntar almacenamiento de bloques continuo a las instancias.
- Pagar sólo por los recursos que realmente se consuma, como las horas de uso de instancias o la transferencia de datos.

### 2.2.1 Características del Servicio

**Elástico**: Amazon EC2 permite aumentar o reducir la capacidad en cuestión de minutos, sin esperar horas ni días. Ofrece la posibilidad de enviar una, cientos o incluso miles de instancias del servidor simultáneamente. Dado que todo está controlado mediante las API del servicio web, la aplicación podrá escalarse automáticamente según aumenten o reduzcan las necesidades.

**Control total**: Siempre se tendrá control total sobre las instancias, obteniendo acceso de usuario raíz a todas ellas, e interactuando con ellas como con cualquier otra máquina. Amazon EC2 ofrece la posibilidad de detener la instancia y mantener los datos en la partición de arranque, para reiniciar a continuación la misma instancia a través de las API del servicio web. Las instancias se pueden reiniciar de forma remota mediante las

API del servicio web. Asimismo, se tiene acceso a la emisión de consola de sus instancias.

**Flexible**: Amazon EC2 nos posee la opción de varios tipos de instancias, sistemas operativos y paquetes de software. Permite seleccionar una configuración de memoria, CPU, almacenamiento de instancias y el tamaño de la partición de arranque óptimo para el sistema operativo y su aplicación. Entre sus opciones de sistemas operativos se incluyen, por ejemplo, varias distribuciones de Linux, Microsoft Windows Server y OpenSolaris. Amazon EC2 está pensado para su uso con otros Amazon Web Services. Trabaja con Amazon Simple Storage Service (Amazon S3), Amazon SimpleDB y Amazon Simple Queue Service (Amazon SQS) para proporcionar una solución completa de computación, procesamiento de consultas y almacenamiento.

**Fiable**: Amazon EC2 ofrece un entorno muy fiable en el que las instancias de sustitución se pueden enviar con rapidez y anticipación. El servicio se ejecuta en los centros de datos y la infraestructura de red acreditados de Amazon. El compromiso del contrato a nivel de servicio de Amazon EC2 es de una disponibilidad del 99,95% en cada Región de Amazon EC2.

**Seguro**: Amazon EC2 ofrece diversos mecanismos para proteger los recursos informáticos. Incluye interfaces de servicio web para configurar el cortafuegos que controla el acceso de red a grupos de instancias, y el acceso entre estos. Al iniciar recursos de Amazon EC2 en Amazon Virtual Private Cloud (Amazon VPC), se pueden aislar las instancias informáticas especificando el rango de IP que se desea utilizar, así como conectarse a su infraestructura de TI existente mediante la red cifrada IPsec VPN estándar del sector.

**Completamente Controlado**: Se tiene control total de las instancias. Se tiene acceso al usuario "root" a cada una y se puede interactuar con estas como si estuvieras en cualquier máquina. Se puede detener la instancia mientras se retiene la información en la partición de arranque y subsecuentemente reiniciar la misma instancia usando el servicio web API.

Amazon EC2 ofrece diversos y potentes elementos para crear aplicaciones de primer nivel, escalables y resistentes a errores, incluyendo:

- **Amazon Elastic Block Store** (EBS): ofrece almacenamiento continuo para instancias de Amazon EC2. Los volúmenes Amazon EBS ofrecen un almacenamiento fuera de la instancia que persiste con independencia de la vida de una instancia. Las características de éstos volúmenes es su fiabilidad y su gran disponibilidad. Se pueden utilizar como particiones de arranque de la instancia de Amazon EC2 o bien conectarse a una instancia de Amazon EC2 en ejecución como dispositivo de bloques estándar. Si se utiliza como partición de arranque, las instancias de Amazon EC2 se pueden detener y reiniciar, lo que permite pagar solo por los recursos de almacenamiento que se utilizan, mientras mantiene el estado de su instancia. Los volúmenes de Amazon EBS ofrecen una duración muy mejorada con respecto a los almacenes de instancias locales de Amazon EC2, ya que los volúmenes de Amazon EBS se replican automáticamente en segundo plano, en una única Zona de disponibilidad. Para aquellos usuarios que deseen contar con una mayor duración, Amazon EBS proporciona la capacidad de crear instantáneas

puntuales coherentes de sus volúmenes, que se almacenarán en Amazon S3 y se replicarán automáticamente en distintas Zonas de disponibilidad. Estas instantáneas se pueden utilizar como punto de partida para nuevos volúmenes de Amazon EBS, y pueden proteger los datos para lograr su duración a lo largo del tiempo. Asimismo, se puede compartir con facilidad estas instantáneas con colaboradores y con otros desarrolladores de AWS.

- **Varias ubicaciones**: Amazon EC2 ofrece la posibilidad de colocar instancias en distintas ubicaciones. Las ubicaciones de Amazon EC2 se componen de Regiones y Zonas de disponibilidad. Las Zonas de disponibilidad son regiones diferentes que están diseñadas para estar aisladas de fallos que se produzcan en otras Zonas de disponibilidad, y que proporcionan conectividad de red de baja latencia a otras Zonas de disponibilidad de la misma Región. Al iniciar instancias en Zonas de disponibilidad distintas, puede proteger a sus aplicaciones en caso de error de una única ubicación. Las Regiones están compuestas por una o más Zonas de disponibilidad, están geográficamente dispersas y se encuentran en áreas geográficas o países diferentes. El compromiso del contrato a nivel de servicio de Amazon EC2 es de una disponibilidad del 99,95% en cada Región de Amazon EC2. Actualmente, Amazon EC2 está disponible en cuatro regiones: EE.UU. Este (Norte de Virginia), EE.UU. Oeste (Norte de California), UE (Irlanda) y Asia Pacífico (Singapur).

- **Elastic IP Addresses**: se trata de direcciones IP estáticas diseñadas para la informática dinámica en nube. Una dirección Elastic IP está asociada a su cuenta, no a una instancia concreta, y se puede controlar esta dirección hasta que se decida, explícitamente, liberarla. Al contrario que las direcciones IP estáticas tradicionales, las Elastic IP Addresses permiten disimular los errores en instancias o Zonas de disponibilidad, al reasignar de forma programada sus direcciones IP públicas a cualquier instancia de su cuenta. En lugar de esperar a que un técnico de datos reconfigure o reemplace su host, o bien esperar a que el DNS se propague a todos sus clientes, Amazon EC2 permite solucionar los problemas con su instancia o su software, mediante la reasignación rápida de su dirección Elastic IP a una instancia de sustitución. Además, tiene la opción de configurar el registro DNS inverso de cualquiera de sus direcciones Elastic IP completando un formulario.

- **Amazon Virtual Private Cloud** (VPC): es un puente seguro y sin fisuras entre la infraestructura de TI de una empresa y la nube de Amazon Web Services. Amazon VPC permite a las empresas conectar su infraestructura existente con un conjunto de recursos informáticos aislados de AWS mediante una conexión de VPN (red privada virtual), así como ampliar sus funciones de gestión existentes, como los servicios de seguridad, los cortafuegos y los sistemas de detección de intrusiones para incluir sus recursos de AWS.

- **Amazon CloudWatch**: es un servicio web que proporciona supervisión para los recursos en nube de AWS, empezando por Amazon EC2. Este servicio Web permite visualizar la utilización de recursos, el funcionamiento operativo y los patrones de demanda en general, incluido el uso de CPU, las operaciones de lectura y escritura en disco y el tráfico de red. Para utilizar

Amazon CloudWatch, sólo hay que seleccionar las instancias de Amazon EC2 cuya supervisión se desea. En cuestión de minutos, Amazon CloudWatch empezará a agregar y almacenar datos de supervisión a los que se puede acceder mediante las API de servicio web o las herramientas de línea de mandato.

- **Auto Scaling**: permite escalar automáticamente la capacidad de Amazon EC2, para aumentarla o reducirla, de acuerdo con las condiciones que se definan. Con Auto Scaling, el cliente puede asegurarse que el número de instancias de Amazon EC2 que esté utilizando aumente sin interrupciones durante los picos de demanda, a fin de mantener el rendimiento, y se reduzca automáticamente durante los períodos de calma en la demanda para minimizar los costes. Auto Scaling resulta especialmente adecuado para aquellas aplicaciones que muestran variaciones de uso según la hora, el día o la semana. Auto Scaling está disponible a través de Amazon CloudWatch.

- **Elastic Load Balancing**: distribuye automáticamente el tráfico entrante de las aplicaciones entre varias instancias de Amazon EC2. Permite conseguir una mayor tolerancia a fallos en las aplicaciones, al proporcionar la capacidad de equilibrio de carga necesaria como respuesta al tráfico entrante de aplicaciones. Elastic Load Balancing detecta instancias en mal estado dentro de un conjunto y redirige automáticamente el tráfico hacia las instancias que se encuentran en buen estado, hasta que se restauran las instancias en mal estado. Es posible habilitar Elastic Load Balancing en una única Zona de disponibilidad o en varias zonas distintas a la vez, para obtener un rendimiento más uniforme de la aplicación al que se le aplica esta funcionalidad.

- **Clústeres High Performance Computing** (HPC): los clientes con cargas de trabajo informáticas complejas, como los procesos paralelos estrechamente asociados, o con aplicaciones afectadas por el rendimiento de red, pueden lograr un alto rendimiento informático y de red proporcionado por la infraestructura personalizada y, al mismo tiempo, beneficiarse de la elasticidad, la flexibilidad y las ventajas de precio de Amazon EC2. Las instancias de tipo Cluster Compute Instances se han diseñado específicamente para proporcionar una funcionalidad de red de alto rendimiento y se pueden iniciar, de forma programada en clústeres, lo que permite a las aplicaciones alcanzar el rendimiento de red de baja latencia necesario para la comunicación de nodo a nodo estrechamente asociada. Las instancias de tipo Cluster Compute Instances proporcionan también un rendimiento de red mejorado, lo que las hace adecuadas para las aplicaciones personalizadas que necesitan realizar operaciones con un alto consumo de red.

### 2.2.2 Tipos de Instancias

Amazon Elastic Compute Cloud permite elegir entre diversos tipos de instancias, para que el cliente pueda elegir aquella que mejor se ajuste a sus requisitos. Cada instancia proporciona una cantidad planificada de capacidad de cálculo y se factura por

hora de uso. Las instancias de Amazon EC2 están agrupadas en cinco familias: Estándar, Micro, Memoria Alta, CPU Alta y Cálculo en Clúster.

A la hora de elegir el tipo de instancia, hay que tener en cuenta las características de la aplicación con respecto a utilización de recursos y seleccionar la familia de instancias y el tamaño óptimos. Para saber qué instancia funcionará mejor para una aplicación, lo mejor es iniciar una instancia y realizar comparaciones para una misma aplicación. Una de las ventajas de EC2 es que se paga por horas de uso, lo que puede resultar cómodo y asequible para probar el rendimiento de la aplicación en distintos tipos y familias de instancias. Por tanto, un buen método para determinar la familia y el tipo de instancia más adecuados consiste en iniciar instancias de prueba y comparar el rendimiento de la aplicación.

El tipo Instancia pequeña equivale al tipo de instancia original de Amazon EC2 que está disponible desde la salida al mercado de Amazon EC2. Actualmente este tipo de instancia es el valor predeterminado para todos los clientes. Si desean utilizar otros tipos de instancias, los clientes deben solicitarlos específicamente usando el panel de control de la API.

Amazon EC2 utiliza diversas medidas para proporcionar a cada instancia la cantidad necesaria y previsible de CPU. A fin de facilitar a los desarrolladores la tarea de comparar capacidad de CPU entre distintos tipos de instancias, Amazon define la Unidad de Cálculo EC2. La cantidad de CPU asignada a una determinada instancia se expresa en términos de estas Unidades de cálculo de EC2. Para gestionar la coherencia y la capacidad de previsión del rendimiento de una Unidad de cálculo de EC2 hay definidos distintos sistemas de comparación y pruebas. Una Unidad de cálculo de EC2 proporciona la capacidad de CPU equivalente a 1,0-1,2 GHz de un procesador 2007 Opteron o 2007 Xeon. Esto también equivale a 1,7 GHz del procesador 2006 Xeon anterior. Es posible que, con el tiempo, se incorporen o sustituyan medidas que forman parte de la definición de una Unidad de cálculo de EC2, en el caso de encontrar medidas que ofrezcan una imagen más concreta de la capacidad de cálculo.

Los distintos tipos de instancias ofrecen un rendimiento mínimo superior o inferior de los recursos compartidos, en función de su tamaño. Cada uno de los tipos de instancia tiene un indicador de rendimiento de E/S (bajo, moderado o alto). Los tipos de instancias con un rendimiento de E/S alto tienen una asignación mayor de recursos compartidos. La asignación de una mayor cantidad de recursos compartidos también reduce la varianza del rendimiento de E/S. Para muchas aplicaciones, un rendimiento de E/S bajo o moderado resulta suficiente. Sin embargo, para aquellas aplicaciones que requieran un rendimiento de E/S mayor o más constante, es necesario considerar la posibilidad de asignarles instancias con rendimiento de E/S alto.

Las instancias estándares tienen una proporción entre memoria y CPU que resulta adecuada para la mayoría de las aplicaciones. Dentro de las instancias estándar podemos elegir entre pequeña, grande y extra grande. Las unidades de cálculos son por cada núcleo virtual.

| Máquina | Núcleos Virtuales | Unidad cálculo | Memoria | Almacenamiento | Plataforma | Rendimiento E/S |
|---|---|---|---|---|---|---|
| Pequeña(defecto) | 1 | 1 | 1,7 GB | 160 GB | 32 bits | Moderado |
| Grande | 2 | 2 | 7,5 GB | 850 GB | 64 bits | Alto |
| Extra-grande | 4 | 2 | 15 GB | 1690 GB | 64 bits | Alto |

Tabla 2.1: Instancias estándares.

Las instancias de memoria alta ofrecen memoria de mayor tamaño para aplicaciones de alto rendimiento, incluidas las aplicaciones de colocación en caché de memoria y de bases de datos. Dentro de este tipo de instancias podemos elegir entre extra grande, extra grande doble y extra grande cuádruple.

| Máquina | Núcleos Virtuales | Unidad cálculo | Memoria | Almacenamiento | Plataforma | Rendimiento E/S |
|---|---|---|---|---|---|---|
| Extra-grande | 2 | 3,25 | 17.1 GB | 420 GB | 64 bits | Moderado |
| Extra-grande doble | 4 | 3,25 | 34,2 GB | 850 GB | 64 bits | Alto |
| Extra-grande cuádruple | 8 | 3,25 | 68,4 GB | 1690 GB | 64 bits | Alto |

Tabla 2.2 Instancias memoria alta.

Las instancias Micro ofrecen una pequeña cantidad de recursos de CPU y permiten ampliar la capacidad de CPU cuando hay nuevos ciclos disponibles. Resultan adecuadas para aplicaciones de bajo rendimiento y para sitios web que consuman ciclos de cálculo significativos de forma periódica. Las instancias Micro ofrecen una pequeña cantidad de recursos de CPU y le permiten ampliar la capacidad de CPU cuando hay nuevos ciclos disponibles. Resultan adecuadas para aplicaciones de bajo rendimiento y para sitios web que consuman ciclos de cálculo significativos de forma periódica. En estado fijo, las instancias Micro reciben una fracción de los recursos de cálculo que reciben las instancias pequeñas. Por lo tanto, si la aplicación ejecuta mucho cálculo o está en un estado fijo, es recomendable utilizar una instancia pequeña o mayor. Sin embargo, las instancias Micro pueden ampliarse periódicamente hasta un máximo de 2 unidades de cálculo, durante cortos periodos de tiempo. Esto supone el doble de unidades de cálculo que las disponibles en una instancia pequeña estándar. Por lo tanto, para una aplicación de rendimiento relativamente bajo o un sitio web con un requisito ocasional de consumo de ciclos de cálculo significativos, es mejor utilizar micro instancias.

| Máquina | Núcleos Virtuales | Unidad cálculo | Memoria | Almacenamiento | Plataforma | Rendimiento E/S |
|---|---|---|---|---|---|---|
| Micro | 1 | 1 ó 2 | 613 MB | EBS | 32 ó 64 bits | Bajo |

Tabla 2.3: Micro instancia

Las instancias de CPU alta tienen proporcionalmente, más recursos de CPU que memoria (RAM) y resultan adecuadas para aplicaciones que requieren gran cantidad de cálculo. Dentro de este tipo de instancias podemos elegir entre media y extra grande.

| Máquina | Núcleos Virtuales | Unidad cálculo | Memoria | Almacenamiento | Plataforma | Rendimiento E/S |
|---|---|---|---|---|---|---|
| Media | 2 | 2,5 | 1,7 GB | 350 GB | 32 bits | Moderado |
| Extra-grande | 8 | 2,5 | 7 GB | 1690 GB | 64 bits | Alto |

Tabla 2.4: instancias de CPU alta

Las instancias de cálculo en clúster ofrecen, en proporción, gran cantidad de recursos de CPU y un rendimiento de red mejorado, lo que las convierte en las más adecuadas para aplicaciones HPC (cálculo de alto rendimiento) y otras aplicaciones exigentes vinculadas a la red. Las instancias de cálculo en clúster tienen un rendimiento de E/S muy alto cuando utilizan 10 Gigabit Ethernet para obtener un alto rendimiento de red y una menor latencia de red entre clústeres.

| Máquina | Núcleos Virtuales | Unidad cálculo | Memoria | Almacenamiento | Plataforma | Rendimiento E/S |
|---|---|---|---|---|---|---|
| Extra-grande cuádruple | 8 | 33,5 | 23 GB | 1690 GB | 64 bits | Muy Alto |

Tabla 2.5: instancias de cálculo en cluster. 33,5 unidades de cálculo EC2 (2 x Intel Xeon X5570, arquitectura "Nehalem" de cuatro núcleos).

Amazon EC2 proporciona instancias de servidor virtualizadas. Algunos recursos, como CPU, memoria y almacenamiento de la instancia, están dedicados a una determinada instancia, otros recursos, como la red y el subsistema de disco, se comparten entre instancias. Si cada instancia de un host físico intenta utilizar la cantidad máxima posible de uno de estos recursos compartidos, cada una de las instancias recibirá la misma cantidad de dicho recurso. Sin embargo, cuando un recurso está infrautilizado, generalmente podrá consumir una mayor parte de dicho recurso mientras esté disponible.

Toda instancia lleva asociada una AMI (Imágenes de Máquina de Amazon). Las AMI de los sistemas operativos han sido pre-configuradas con una lista de sistemas operativos cada vez mayor. Entre los sistemas operativos que actualmente se pueden utilizar con instancias de Amazon EC2 se encuentran los siguientes:

| Ubuntu Linux | Debian | Fedora | Open Solaris |
|---|---|---|---|
| Windows Server 2003/2008 | Amazon Linux AMI | Oracle Enterprise Linux | SUSE Linux Enterprise |
| Gentoo Linux | Red Hat Enterprise Linux | | |

Tabla 2.6: Imágenes disponibles para máquinas de Amazon.

Amazon permite utilizar herramientas de empaquetado para cargar sistemas operativos propios del cliente. Además Amazon EC2 permite a los clientes crear y

personalizar imágenes de máquina de Amazon (AMI) con el software necesario. Para ello, ofrece[21]:

- Entornos de desarrollo de aplicaciones (IBM sMash, Ruby on Rails).
- Servidores de aplicaciones (Java Aplication Server, Oracle WebLogic Server, IBM WebSphere Application Server).
- Codificadores de video y Streaming (Wowza Media Server Pro, Windows Media Server), etc.

### 2.2.3 Opciones de compra de instancias Amazon EC2

Los precios se basan en la región en la que se ejecuta la instancia. A día de hoy Amazon EC2 ofrece sus servicios desde cuatro regiones diferentes, Virginia del Norte en EE.UU, California del Norte en EE.UU, Irlanda en la Unión Europea, APAC en Singapur.

Para escoger fácilmente el número, el tamaño y la configuración de las instancias informáticas, Amazon EC2 proporciona a los clientes tres modelos de compra diferentes que proporcionan la flexibilidad necesaria para optimizar costes, instancias bajo demanda, instancias reservadas e instancias puntuales.

Con las instancias bajo demanda ("On-Demand Instances") se paga una tarifa fija por la capacidad informática por hora, sin ningún tipo de compromiso a largo plazo. Esto liberará de los costes y las complejidades de la planificación, la compra y el mantenimiento del hardware y transformará lo que normalmente son grandes costes fijos en costes variables mucho más pequeños. Con las instancias bajo demanda también se elimina la necesidad de comprar una "red de seguridad" de capacidad para gestionar picos de tráfico periódicos. Amazon EC2 siempre se esfuerza por tener suficiente capacidad bajo demanda disponible para resolver todas las necesidades, pero durante periodos de muy alta demanda, es posible que no se puedan lanzar determinados tipos de instancia bajo demanda en zonas de disponibilidad concretas durante breves periodos de tiempo.

Las instancias bajo demanda están recomendadas para:

- Aquellos usuarios que desean acceder al bajo coste y a la flexibilidad de Amazon EC2 sin pagos por adelantado ni compromisos a largo plazo.
- Aplicaciones con carga de trabajo a corto plazo, con picos frecuentes o impredecibles que no pueden interrumpirse.
- Aplicaciones que se están desarrollando o probando en Amazon EC2 por primera vez.

[21] http://aws.amazon.com/es/ec2/#os

| | | EE.UU.: Virginia del Norte | | EE.UU.: California del Norte | | UE: Irlanda | | APAC: Singapur | |
|---|---|---|---|---|---|---|---|---|---|
| Familia | Instancia | Linux | WINDOWS | Linux | WINDOWS | Linux | WINDOWS | Linux | WINDOWS |
| estándar | Pequeño | $ 0.085 | $ 0.12 | $ 0.095 | $ 0.13 | $ 0.095 | $ 0.13 | $ 0.095 | $ 0.13 |
| | Grande | $ 0.34 | $ 0.48 | $ 0.38 | $ 0.52 | $ 0.38 | $ 0.52 | $ 0.38 | $ 0.52 |
| | Extragrande | $ 0.68 | $ 0.96 | $ 0.76 | $ 1.04 | $ 0.76 | $ 1.04 | $ 0.76 | $ 1.04 |
| micro | Micro | $ 0.02 | $ 0.03 | $ 0.025 | $ 0.035 | $ 0.025 | $ 0.035 | $ 0.025 | $ 0.035 |
| alta memoria | Extragrande | $ 0.50 | $ 0.62 | $ 0.57 | $ 0.69 | $ 0.57 | $ 0.69 | $ 0.57 | $ 0.69 |
| | Doble extragrande | $ 1.00 | $ 1.24 | $ 1.14 | $ 1.38 | $ 1.14 | $ 1.38 | $ 1.14 | $ 1.38 |
| | Cuádruple Extragrande | $ 2.00 | $ 2.48 | $ 2.28 | $ 2.76 | $ 2.28 | $ 2.76 | $ 2.28 | $ 2.76 |
| alta potencia | Mediano | $ 0.17 | $ 0.29 | $ 0.19 | $ 0.31 | $ 0.19 | $ 0.31 | $ 0.19 | $ 0.31 |
| | Extragrande | $ 0.68 | $ 1.16 | $ 0.76 | $ 1.24 | $ 0.76 | $ 1.24 | $ 0.76 | $ 1.24 |
| clústeres | Cuádruple Extragrande | $ 1.60 | N/D* | N/D* | N/D* | N/D* | N/D* | N/D* | N/D* |

Tabla 2.7: Precios de las instancias bajo demanda. *Actualmente Windows no se encuentra disponible para Instancias de clústeres.

Con las instancias reservadas (“Reserved Instances”) se tiene la opción de abonar una tarifa única reducida para cada instancia que se desee reservar, durante uno o tres años, y obtener un importante descuento en la tarifa horaria por uso de dicha instancia. Tras el pago único por una instancia, la instancia queda reservada, sin ninguna otra obligación, pudiendo optar por ejecutar la instancia por la tarifa horaria reducida mientras dure su plazo o bien por no pagar la tarifa de uso mientras no se utilice la instancia. Amazon EC2 garantizará que la Instancia reservada siempre esté disponible para el sistema operativo (ej. Linux/UNIX o Windows) y la zona de disponibilidad en la que la adquirió. Para aquellas aplicaciones que presentan necesidades de estado continuas, las Instancias reservadas pueden proporcionar ahorros de casi el 50% en comparación con la utilización de instancias bajo demanda. En términos funcionales, las instancias reservadas y las instancias bajo demanda funcionan de forma idéntica.

Las instancias reservadas están recomendadas para:

- Aplicaciones con un estado constante o con uso predecible.
- Aplicaciones que necesitan capacidad reservada, incluyendo recuperación de desastres.
- Usuarios que tienen la posibilidad de realizar pagos por adelantado para reducir aún más los costes informáticos.

EE.UU.: Virginia del Norte

| | | Tarifa Única | | | |
|---|---|---|---|---|---|
| Familia | Instancia | 1 Año | 3 Años | Linux/UNIX | WINDOWS |
| estándar | Pequeño | $ 227.50 | $ 350 | $ 0.03 | $ 0.05 |
| | Grande | $ 910 | $ 1400 | $ 0.12 | $ 0.20 |
| | Extragrande | $ 1820 | $ 2800 | $ 0.24 | $ 0.40 |
| micro | Micro | $ 54 | $ 82 | $ 0.007 | $ 0.013 |
| alta memoria | Extragrande | $ 1325 | $ 2000 | $ 0.17 | $ 0.24 |
| | Doble extragrande | $ 2650 | $ 4000 | $ 0.34 | $ 0.48 |
| | Cuádruple Extragrande | $ 5300 | $ 8000 | $ 0.68 | $ 0.96 |
| alta potencia | Mediano | $ 455 | $ 700 | $ 0.06 | $ 0.125 |
| | Extragrande | $ 1820 | $ 2800 | $ 0.24 | $ 0.50 |
| clústeres | Cuádruple Extragrande | $ 4290 | 6590 | $ 0.56 | N/D |

Tabla 2.8: precios instancias reservadas en EE.UU.: Virginia del Norte

EE.UU.: California del Norte, UE: Irlanda, APAC: Singapur

| | | Tarifa Única | | | |
|---|---|---|---|---|---|
| Familia | Instancia | 1 Año | 3 Años | Linux/UNIX | WINDOWS |
| estándar | Pequeño | $ 227.50 | $ 350 | $ 0.04 | $ 0.06 |
| | Grande | $ 910 | $ 1400 | $ 0.16 | $ 0.24 |
| | Extragrande | $ 1820 | $ 2800 | $ 0.32 | $ 0.48 |
| micro | Micro | $ 54 | $ 82 | $ 0.01 | $ 0.016 |
| alta memoria | Extragrande | $ 1325 | $ 2000 | $ 0.24 | $ 0.32 |
| | Doble extragrande | $ 2650 | $ 4000 | $ 0.48 | $ 0.64 |
| | Cuádruple Extragrande | $ 5300 | $ 8000 | $ 0.96 | $ 1.28 |
| alta potencia | Mediano | $ 455 | $ 700 | $ 0.08 | $ 0.145 |
| | Extragrande | $ 1820 | $ 2800 | $ 0.32 | $ 0.58 |
| clústeres | Cuádruple Extragrande | N/D* | N/D* | N/D* | N/D |

Tabla2.9: precios instancias reservadas en las regiones de. EE.UU.: California del Norte, UE: Irlanda y APAC: Singapur

Las instancias puntuales (“Spot Instances”) ofrecen la posibilidad de que el cliente compre capacidad informática sin ningún tipo de compromiso por adelantado, y a unas tarifas por hora que suelen ser más bajas que las que corresponden a la tarifa bajo

demanda. Las instancias puntuales permiten especificar el precio por hora máximo que el cliente está dispuesto a pagar para ejecutar un tipo de instancia determinado. Amazon EC2 establece un precio puntual para cada tipo de instancia de cada región, que será el precio que todos los clientes pagarán para ejecutar una instancia puntual durante ese periodo determinado. El precio cambia periódicamente según la oferta y la demanda, pero los clientes nunca pagarán más que el precio máximo que hayan especificado. Si el precio puntual supera el precio máximo de un cliente, Amazon EC2 finalizará la instancia del cliente. Aparte de estas diferencias, el comportamiento de las instancias puntuales es exactamente el mismo que el de las Instancias bajo demanda o reservadas.

Las instancias puntuales están recomendadas para:

- Aplicaciones que tienen hora de inicio y de fin flexibles.
- Aplicaciones que únicamente son viables a muy bajos precios informáticos.
- Usuarios con necesidades informáticas urgentes de grandes cantidades de capacidad adicional.

| | | EE.UU.: Virginia del Norte | | EE.UU.: California del Norte | | UE: Irlanda | |
|---|---|---|---|---|---|---|---|
| Familia | Instancia | Linux | WINDOWS | Linux | WINDOWS | Linux | WINDOWS |
| estándar | Pequeño | $ 0.031 | $ 0.052 | $ 0.041 | $ 0.068 | $ 0.033 | $ 0.066 |
| | Grande | $ 0.115 | $ 0.119 | $ 0.161 | $ 0.261 | $ 0.167 | $ 0.277 |
| | Extragrande | $ 0.25 | $ 0.403 | $ 0.307 | $ 0.534 | $ 0.305 | $ 0.552 |
| micro | Micro | $ 0.007 | $ 0.013 | $ 0.01 | $ 0.016 | $ 0.01 | $ 0.016 |
| alta memoria | Extragrande | $ 0.167 | $ 0.223 | $ 0.237 | $ 0.316 | $ 0.245 | $ 0.329 |
| | Doble extragrande | $ 0.415 | $ 0.549 | $ 0.582 | $ 0.701 | $ 0.547 | $ 0.742 |
| | Cuádruple Extragrande | $ 0.867 | $ 1.118 | $ 1.167 | $ 1.406 | $ 1.072 | $ 1.407 |
| alta potencia | Mediano | $ 0.061 | $ 0.13 | $ 0.08 | $ 0.16 | $ 0.077 | $ 0.167 |
| | Extragrande | $ 0.25 | $ 0.507 | $ 0.314 | $ 0.682 | $ 0.312 | $ 0.669 |

Tabla 2.10: precios instancias puntuales. Actualmente, Windows no se encuentra disponible para Instancias para clústeres.

# 2.3 Otros Clouds públicos en la red

Varias son las grandes empresas que se han dedicado a ofrecer de servicios de computación en la Nube, promoviendo el fácil acceso a nuestra información, los bajos costos, la escalabilidad y muchas características que nos hace pensar en la comodidad que nos brindan, entre ellas podemos mencionar:

- **Google Apps Engine**[22]: es un Servicio basado en Infraestructura SaaS. App Engine es un servicio de alojamiento de páginas web que presta Google de forma gratuita hasta determinadas cuotas, este servicio permite ejecutar aplicaciones sobre la infraestructura de Google, proporcionando dominios propios para las aplicaciones con la siguiente estructura: '<midominio>.appspot.com'. Por el momento las cuentas gratuitas tienen un límite de 1 Gigabyte de almacenamiento permanente y la suficiente cantidad de ancho de banda y CPU para 5 millones de páginas vistas mensuales, y si la aplicación supera estas cuotas, se pueden comprar cuotas adicionales por un bajo costo. Actualmente las aplicaciones se implementan mediante los lenguajes de programación Python y Java. Google App Engine incluye las siguientes funciones:

  - Servidor web dinámico, totalmente compatible con las tecnologías web más comunes.
  - Almacenamiento permanente con funciones de consulta, orden y transacciones,
  - Escalado automático y balanceo de carga.
  - API para autenticar usuarios y enviar correo electrónico a través de las cuentas de Google.
  - Un completo entorno de desarrollo local que simula Google App Engine en tu equipo.
  - Tareas programadas para activar eventos en momentos determinados y en intervalos regulares.

Otros de los servicios de Google en la Nube, es Google Apps, que brinda el servicio de herramientas para empresas basadas en web fiables y seguras como Gmail, Google Talk, Google Calendar y Google Docs, etc.

- **Azure de Microsoft**[23]: es una Plataforma ofrecida como servicio (PaaS) alojada en los centros de procesamiento de datos de Microsoft.

Entre los distintos servicios de Windows Azure cabe destacar:

- Cómputo: proporciona el alojamiento de las aplicaciones.
- Almacenamiento: persistente y duradero en la Nube a través de cuatro servicios básicos.
- Bases de datos: altamente disponibles y escalables en la nube.
- Máquinas virtuales: dedicadas con Windows Server 2008 R2. Dependiendo de las necesidades de la aplicación se pueden seleccionar diferentes perfiles de máquina: número de procesadores, tamaño de la memoria, espacio en disco, etc.
- Red de entrega: proporcionando un gran ancho de banda a través de 24 nodos físicos globales.
- Almacenamiento en cache: distribuido en memoria caché de la aplicación de servicio.
- Virtual Network: funcionalidad para conectar redes en las instalaciones y aplicaciones en la Nube.
- Service Bus: mensajería segura para las aplicaciones distribuidas e híbridos.
- Control de Acceso: Basada en estándares de servicios de identidad y control de acceso.
- Business Intelligence: Desarrollar e implementar informes operativos en la Nube

[22] http://code.google.com/intl/es-ES/appengine/
[23] http://www.microsoft.com/windowsazure/

mediante herramientas familiares.

- **IBM Cloud Computing**[24]: engloba los tres tipos de infraestructura como servicio. Provee infraestructura como servicio (IaaS) mediante IBM SmartCloud Enterprise, plataforma como servicio (PaaS) a través de IBM Cloud Service Provider Platform y software como servicio (SaaS) por medio de IBM WebSphere. IBM SmartCloud Enterprise es la infraestructura en la Nube como servicio (IaaS) de IBM, diseñado para proporcionar un acceso rápido y seguro a entornos de servidores virtuales. Ofrece un soporte "Premium" avanzado, sectores de conectividad VLAN público y privado, gran variedad de instancias las cuales se pagan sólo por uso, paquetes de software, almacenamiento persistente, así como la personalización de imágenes privada de máquinas virtuales adjuntando una configuración adicional o paquetes de software de instalación.

Una de las grandes dificultades que se le puede presentar a cualquier usuario es la elección de uno de estos proveedores. La elección más acertada debería estar basada tanto en el tipo de servicio que se necesite (IaaS, PaaS, SaaS), como en el desembolso económico y el rendimiento computacional ofrecido.

Para el caso de infraestructura como servicio, que es el tipo de servicio para el desarrollo de este proyecto, es necesario evaluar la capacidad computacional de las instancias como el coste de las mismas. La opción más adecuada sería ejecutar pequeñas pruebas, "benchmarks" personalizados, averiguando que proveedor Cloud se ajusta más a nuestras necesidades. Pero ésta tarea puede resultar muy complicada o casi imposible debido a la gran oferta de servicios Cloud. Otra forma más sencilla sería basarse en las diferentes pruebas ya realizadas a estos proveedores [DAMI].

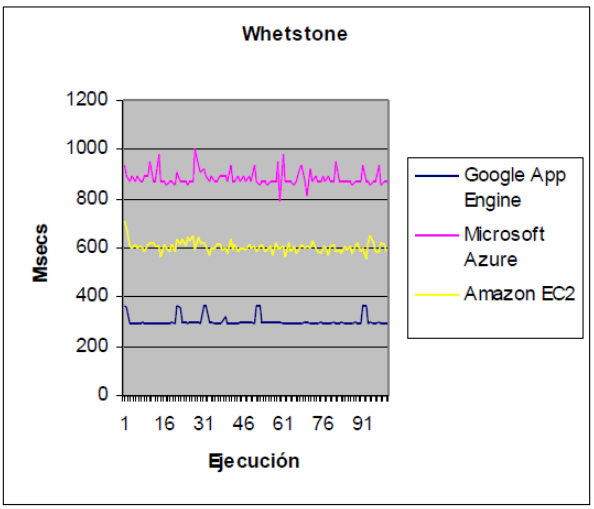


Figura 2.4: Gráfico comparativo usando el benchmark Whetstone, en donde se muestran los tiempos de ejecución.

Dado que la comparación de rendimiento computacional puede no encontrarse o no haberse realizado sobre los proveedores de servicios que queremos contratar, se puede elegir aquel proveedor sólo según su coste económico.

[24] http://www-05.ibm.com/es/cloudcomputing/

Una comparación, a nivel IaaS, de los tipos de máquinas y precios ofertados por Windows Azure y Amazon EC2 puede verse en la siguiente tabla.

Microsoft

| Tipo Instancia | CPU | Memoria | Almacenamiento | Precio/ hora |
|---|---|---|---|---|
| Extra Small | 1 GHz | 768 MB | 20 GB | $ 0.05 |
| Small | 1.6 GHz | 1.75 GB | 225 GB | $ 0.12 |
| Medium | 2·1.6 GHz | 3.5 GB | 490 GB | $ 0.24 |
| Large | 4·1.6 GHz | 7 GB | 1000 GB | $ 0.48 |
| ExtraLarge | 8·1.6 GHz | 14 GB | 2040 GB | $ 0.96 |

amazon.com

| Tipo Instancia | CPU | Memoria | Almacenamiento | Precio/ hora |
|---|---|---|---|---|
| Micro | 2·1,7 GHz | 613 MB | EBS | $ 0.02 |
| Small | 1,7 GHz | 1.7 GB | 160 GB | $ 0.085 |
| Grande | 4·1,7 GHz | 7.5 GB | 850 GB | $ 0.34 |
| ExtraGrande | 8·1,7 GHz | 15 GB | 1690 GB | $ 0.62 |

Figura 2.5: Comparación de precios entre Microsoft y Amazon a nivel PaaS.

Otra comparación a nivel de SaaS, entre Google App Engine, Windows Azure y Amazon EC2.

Google

| Resource | Unit | Unit cost |
|---|---|---|
| Outgoing Bandwidth | gigabytes | $ 0.12 |
| Incoming Bandwidth | gigabytes | $ 0.10 |
| CPU Time | CPU hours | $ 0.10 |
| Stored Data | gigabytes/month | $ 0.15 |
| Recipients Emailed | Recipients | $ 0.0001 |

Microsoft

| Resource | Unit | Unit cost |
|---|---|---|
| Data Transmissions-in | gigabytes | $ 0.10 |
| Incoming Bandwidth | gigabytes | $ 0.15 |
| Storage | gigabytes/month | $ 0.15 |
| Compute Time | machine hours | $ 0.12 |
| Storage Transactions | 10K Application Requests | $ 0.01 |

amazon.com

| Resource | Unit | Unit cost |
|---|---|---|
| Data Transfer-in | gigabytes | $ 0.10 |
| Data Transfer-put | gigabytes | $ 0.14 |
| Storage | CPU hours | $ 0.15 |
| CPU Compute Time | Gigabytes per month | $ 0.125 |

Figura 2.6: Comparación de precios entre Google, Microsoft y Amazon a nivel SaaS.

Para nuestro proyecto se ha optado por el uso de Cloud proporcionado por Amazon, debido a que supone un coste más económico y ofrece una capacidad computacional muy similar a la de otros servidores de Cloud.

# 3. Criptografía: RSA

## 3.1 Criptografía: conceptos fundamentales

La *criptografía* (palabra que proviene de la unión de los términos griegos "κρυμμένο" oculto y "γραφή" escritura) es el estudio de los principios y las técnicas por las cuales la información puede ocultarse en textos cifrados para después ser revelada por usuarios autorizados empleando la clave privada, pero en el que es imposible o inviable computacionalmente para una persona que no esté autorizada para ello. Es decir, la criptografía es un conjunto de técnicas para la protección de la información. Entre las disciplinas que engloba cabe destacar la Teoría de la Información, la Teoría de Números –o Matemática Discreta- y la Complejidad Algorítmica. Conviene hacer notar que la palabra Criptografía sólo hace referencia al uso de códigos, por lo que no engloba a las técnicas que se usan para recuperar dichos códigos sin conocimiento de la clave, conocidas en su conjunto como Criptoanálisis. En cualquier caso ambas disciplinas están íntimamente ligadas y subordinadas al término más general conocido como C*riptología*, quedando pues

$$Criptología \stackrel{\text{def}}{=} Criptografía + Criptoanálisis$$

Un *criptosistema* es una quíntupla *(M, C, K, E, D)*, donde:

- *M* representa el conjunto de todos los mensajes sin cifrar que pueden ser enviados (comúnmente denominado texto plano, texto original o "*plain-text*").
- *C* representa el conjunto de todos los mensajes cifrados (también llamados criptogramas, texto cifrado o "*cipher-text*").
- *K* representa el conjunto de claves que se pueden emplear en el criptosistema, tanto para cifrar como para descifrar.
- *E* es el conjunto de transformaciones de cifrado o familia de funciones que se aplica a cada elemento de *M* para obtener un elemento de *C*. Existe una transformación diferente $E_k$ para cada valor posible de la clave *'k'*.

$$E_k: M \rightarrow C$$

- D es el conjunto de transformaciones de descifrado, análogo a *E*.

$$D_k: C \rightarrow M$$

Todo criptosistema ha de cumplir:

$$D_k\big(E_k(m)\big) = m$$

Es decir, *D* y *E* son un par de funciones invertibles: si tenemos un mensaje *'m'*, lo ciframos empleando la clave *'k'* y luego lo desciframos empleando la misma clave, obtendremos de nuevo el mensaje original *'m'*.

En todos los criptosistemas se tienen que dar las siguientes propiedades:

i) Las transformaciones de cifrado y descifrado han de ser eficientes para cualquiera que sea la clave.
ii) El sistema ha de ser fácil de usar.
iii) La seguridad del sistema sólo debe depender del secreto de las claves, y no del secreto del algoritmo de cifrado, ya que este ha de ser público para que sea analizado por toda la comunidad científica con el fin de encontrar debilidades y hacerlo más robusto.

Existen tres clases fundamentales de criptosistemas:

- Criptosistemas simétricos o de clave privada. Son aquellos que emplean la misma clave *'k'* tanto para cifrar como para descifrar. Presentan el inconveniente de que para ser empleados en comunicaciones la clave *'k'* debe estar tanto en el emisor como en el receptor. Se dividen en dos grandes grupos de cifrado, según su forma de codificar la información:
  - Cifrado en bloque.
  - Cifrado en flujo.

- Criptosistemas asimétricos o de clave pública. Emplean una doble clave $(k_{priv}, k_{pub})$, donde la primera se conoce como clave privada y la segunda como pública. Una de ellas se emplea para la transformación *'E'* de cifrado y la otra para la transformación *D* de descifrado. Estos criptosistemas, como veremos más adelante en detalle, deben cumplir además que el conocimiento de la clave pública no permita calcular la clave privada, o al menos que dicho cálculo sea computacionalmente inviable.

- Criptosistemas híbridos. Combinación de los dos anteriores. Mediante un sistema asimétrico los usuarios intercambian de forma segura la clave que van a usar para codificar y decodificar los datos con un sistema simétrico.

## 3.2 Criptoanálisis

Como ya vimos en la introducción de este capítulo, se denomina 'Criptoanálisis' al conjunto de técnicas que se usan para recuperar los mensajes cifrados sin conocimiento de la clave. Otras personas la describen como la ciencia que estudia los ataques hechos contra esquemas criptográficos.

El objetivo principal de la criptografía es mantener en secreto un texto original, a salvo de personas no autorizadas que intentan obtener la información de dicho texto. Otros posibles atacantes pueden tener otros objetivos, como puede ser por ejemplo modificar el mensaje, comprometiendo la integridad de este. En adelante asumiremos que cualquier adversario que intente atacar un criptosistema tiene acceso total al canal de comunicaciones.

Antes de pasar a describir los principales ataques a esquemas criptográficos vamos a exponer uno de los principios básicos que describió en el siglo XIX A. Kerkhoff para la Criptografía, conocido como '*el principio de Kerkhoff*' [SASA]:

*"La seguridad de un sistema no debe depender en mantener en secreto el algoritmo, sino sólo la clave".*

Por lo que en adelante asumiremos que el atacante tiene total conocimiento del criptosistema, incluyendo cada uno de los algoritmos utilizados y sus respectivas implementaciones. Sus ataques se concentrarán en intentar recuperar textos originales de textos cifrados capturados o, a mayor escala, conseguir la clave secreta. Una tercera vía que tiene el atacante es manipular un mensaje original, como ya dijimos anteriormente.

La clasificación de los ataques de acuerdo a los datos que se requieren, es la siguiente:

- *Ataque a texto cifrado* ("*Ciphertext-only attack*") - El atacante tiene la capacidad de obtener textos cifrados. Cualquier método de cifrado que no sea capaz de resistir un ataque de este tipo se considera totalmente inseguro.
- *Ataque con texto plano conocido* ("*Known-plaintext attack*.") - El atacante tiene la habilidad de conseguir parejas de texto original-texto cifrado y, usando la información que obtiene de esos pares intenta decodificar otro texto cifrado del que no posee texto original.
- *Ataque con texto plano escogido* ("*Chosen-plaintext attack*") - El atacante es capaz de conseguir textos cifrados de los textos originales que él escoja, para intentar decodificar un texto cifrado del que no tiene su texto original asociado. Este tipo de ataque asume que el atacante primero ha de obtener las parejas texto original-texto cifrado y hacer un análisis previo, esto es, sólo necesita acceder una vez al dispositivo de codificación.
- *Ataque basado en texto plano elegido adaptativamente* ("*Adaptively-chosen-plaintext attack*") - Este ataque es similar al anterior, salvo que el atacante puede hacer algún tipo de análisis sobre las parejas de texto original-texto cifrado y posteriormente obtener nueva información en forma de parejas. Según va obteniendo información, puede ir decodificando más pares de texto o analizar lo que ya tiene para obtener más información. Tiene acceso al dispositivo de codificación o puede hacer uso repetido de él.
- *Ataque con texto cifrado escogido y ataque basado en texto cifrado elegido adaptativamente ("Chosen-ciphertext attack & adaptively-chosen-ciphertext attack")* - Estas estrategias de ataque son idénticas que las dos anteriores, cambiando texto original por texto cifrado, esto es, el atacante puede elegir textos cifrados y obtener sus correspondientes textos planos, es decir, tiene acceso al dispositivo de decodificación.

## 3.3 Historia de la Criptología: Cifrados de clave pública/privada

Antes del siglo XX, la Criptología se consideraba un arte vinculado directamente con el poder fáctico, ligado a secretos de estado, asuntos militares, de espionaje y diplomáticos, siempre seguido de una aureola de misterio. Durante el siglo XX se produjo un cambio radical en esta concepción. En la primera mitad del siglo, aunque se

realizaron las primeras aproximaciones científicas importantes, todavía su uso se restringía a ámbitos político-militares: su empleo en las transmisiones de mensajes en las dos Guerras Mundiales cobró una importancia decisiva, originando un gran auge tanto de las técnicas como de las máquinas de cifrar. En la Primera, la ruptura del famoso telegrama *Zimmermann* provocó la entrada de EEUU en la confrontación mundial, hecho crucial que cambió el rumbo de la historia. En la Segunda, los aliados desde casi el inicio fueron capaces de leer los mensajes secretos alemanes ya que habían comprometido su máquina de cifrar, la denominada *Enigma*, con el apoyo del precursor de los ordenadores modernos, el *Colossus* [TUNI].

La Criptología empezó a ser considerada una ciencia en la segunda mitad del siglo XX, cuando dos hechos significativos marcaron un punto de inflexión en su historia. El primero de ellos, los estudios que en el año 1948 realiza Claude Shannon sobre Teoría de la Información y Criptología. Desde ese momento, la Criptología deja de ser considerada como un mero arte rodeado de un cierto halo de misterio y en algunos casos escepticismo, para ser tratada como una rama más de las matemáticas. Por aquel entonces los cifrados eran demasiado complicados y caros para su uso diario, por lo que tendría que esperar hasta la década de los setenta en la cual la potencia computacional se hizo más asequible con la introducción de los microprocesadores [UJAL]. En el año 1976, Whitfield Diffie y Martin Hellman publicaron "New Directions in Cryptography" [DIRE], proponiendo por primera vez la idea y el concepto de Criptografía de clave pública así como de firma digital, ambas como soluciones al intercambio de claves en un medio no seguro basado en el problema del logaritmo discreto. Este hecho asentó las bases de lo que hoy conocemos como Criptografía de clave pública, o comúnmente denominada Criptografía moderna.

Un año después tres integrantes del Instituto de Tecnología de Massachusetts, Ron Rivest, Adi Shamir y Len Adleman, propusieron un criptosistema de clave pública práctico basado en test de primalidad y en factorización de números enteros, comúnmente conocido por las siglas de sus creadores como criptosistema RSA. Basaron dicho sistema en la encriptación y descifrado de enteros en aritmética modular sobre *'n'*, siendo *'n'* el producto de dos números primos grandes.

En la Criptografía, de todos es sabido que existen dos 'universos' paralelos: el primero lo compone la sociedad de investigación académica, con fines públicos y accesibles. El segundo, está formado por todas las agencias de seguridad y ejércitos de los países, cuyas investigaciones son de carácter secreto. Por esta razón, tres investigadores de la agencia de seguridad británica (CESG/GCHQ) Ellis, Cokcs y Williamson reclamaron que ellos habían descubierto el cifrado de clave pública años antes de que Rivest, Shamir, y Adleman lo hicieran, aunque no pudieron publicarlo. Los trabajadores para la agencia británica no hicieron referencia alguna en sus documentos a firma digital, al contrario que los investigadores americanos.

En 1982 Rivest, Shamir y Adleman fundaron la empresa de seguridad 'RSA Data Security' para comercializar su descubrimiento, pero tuvieron que esperar varios años para que sus aplicaciones matemáticas pudieran ser efectivas debido a las prestaciones de los ordenadores de la época. En la actualidad, la empresa ha sido adquirida por la compañía EMC, siendo denominada por ésta como su división de seguridad, presentando productos tanto de seguridad como de firma digital.

# 3.4 RSA: conceptos criptográficos

Hoy en día el sistema RSA es el criptosistema de clave pública más importante y extendido. Su seguridad está estrechamente relacionada con la dificultad de encontrar la factorización de un número entero positivo compuesto el cual es producto de dos números primos grandes. De hecho, el problema de calcular el exponente de decodificación RSA *'d'* a partir de la clave pública y el problema de factorizar '*n'* son computacionalmente equivalentes.

Vamos a proceder a calcular las claves pública y privada. Para ello, generamos dos números enteros primos grandes de forma aleatoria y completamente independiente, a los que llamaremos *'p'* y *'q'*. Ahora calculamos su producto

$$n = p \times q$$

Ahora elegimos un número entero *e* de forma que se cumpla

$$1 < e < \varphi(n) = (p-1)(q-1)\ y\ mcd\big(e,(p-1)(q-1)\big) = 1 \qquad (Ec.3.1)$$

Donde *'e'* es siempre impar ya que $p-1$ es par. Después, calculamos un cierto número entero *'d'* que cumple

$$1 < d < (p-1)(q-1)\ y\ de \equiv 1\ mod\ ((p-1)(q-1))$$

Como $mcd\big(e,(p-1)(q-1)\big) = 1$, el número *'d'* existe y puede ser calculado con el Algoritmo de Euclides Extendido [BUCH, capítulo 1.9].

Con esto ya hemos calculado nuestras claves RSA: nuestra clave privada es *'d'*, al que llamaremos 'exponente de descifrado' y la pública la forman el par $(n, e)$, donde el número *'n'* se llama 'módulo RSA' y *'e'* es conocido como 'exponente de cifrado'. Observamos que la clave secreta puede ser calculada a partir del exponente de cifrado si los factores *'p'* y *'q'* son conocidos, por lo que si el atacante es capaz de averiguar la factorización en números primos de *'n'* entonces puede averiguar nuestra clave secreta.

Una vez generadas las claves, podemos proceder a publicar nuestra clave pública de forma que cualquiera pueda comunicarse con nosotros de modo seguro. Supongamos que otra persona, llamémosla Amigo, quiere enviarnos un mensaje confidencial. Vamos a explicar cómo cifrar y descifrar números con un criptosistema RSA, para después mostrar cómo puede ser usado como un cifrado en bloques.

Para codificar números con RSA, primero definimos el conjunto $M \stackrel{\text{def}}{=}$ espacio de mensajes a cifrar. Éste está formado por todos los enteros *'m'* tales que $0 \leq m < n$. Entonces *'m'* se codifica según la siguiente ecuación:

$$c = m^e (mod\ n)$$

Consiguiendo el mensaje cifrado *'c'*. Por lo tanto si nuestro Amigo conoce la clave pública $(n, e)$, puede codificar el mensaje que quiere enviarnos. Para que este proceso

sea eficiente, Amigo utilizará el Algoritmo de Exponenciación Rápida [BUCH, capítulo 2.12].

El descifrado de RSA se basa en la Fórmula de Euler, el cual nos dice que para dos números cualesquiera *'l'* y *'a'* con $mcd(l,a) = 1$ se tiene que $a^{\varphi(l)} \equiv 1(mod\ l)$. Como *'m'* es primo con *'n'* tenemos que $m^{\varphi(n)} \equiv 1\ (mod\ n)$. Además, como hemos visto antes,

$$d \times e \equiv 1\ (mod\ \varphi(n)) \rightarrow\ \varphi(n) | (d \times e - 1) \rightarrow e \times d = 1 + k \times \varphi(n), para\ algún\ entero\ k$$

Esto nos lleva al método para obtener *'m'*:

$$c^d \equiv (m^e)^d \equiv m^{e \times d} \equiv m^{1+k \times \varphi(n)} \equiv m \times \left(m^{\varphi(n)}\right)^k \equiv m\ (mod\ n) \qquad (Ec.3.2)$$

Esta propiedad nos muestra que RSA es un criptosistema por definición: para cada función de codificación existe una función de decodificación.

Ahora vamos a ver cómo utilizar este sistema como un cifrado en bloques. Consideremos $k = \lfloor \log_N n \rfloor$, siendo *N* un número entero positivo, la longitud de bloque de texto plano a codificar. Usamos el siguiente alfabeto $\Sigma = \mathbb{Z}_N = \{0, 1, \dots, N-1\}$ donde *N* es la base del sistema de numeración al que pertenece una cierta palabra $m_1 \dots m_k \in \Sigma^k$. Dicha palabra corresponde al entero:

$$m = \sum_{i=1}^{k} m_i \times N^{k-i}$$

Donde *'m'* es el texto claro entero a codificar. Ahora nuestro Amigo procede a identificar y relacionar los bloques de $\Sigma^k$ con sus correspondientes números enteros. El bloque *'m'* se cifra resolviendo la siguiente ecuación $c = m^e (mod\ n)$, obteniendo el texto cifrado *'c'*.

El entero *'c'* está representado en base N. Como $0 \leq c < n < N^{k+1}$, la expansión N-ádica de *'c'* tiene una longitud máxima de $k + 1$. Por lo tanto, podemos escribirlo de la forma:

$$c = \sum_{i=0}^{k} c_i\, N^{k-i}\,, c_i \in \Sigma, 0 \leq i \leq k$$

El texto cifrado es, por lo tanto, $c = c_0 c_1 \dots c_k$. De esta forma, RSA mapea bloques de longitud *'k'* de forma inyectiva en bloques de longitud $k + 1$. Aunque esto no es por definición un cifrado en bloque (ya que la longitud de tanto los bloques de mensajes a cifrar como los bloques de mensajes cifrados deberían de ser la misma), se puede utilizar para ello.

Una vez completado el proceso de codificación en bloque, procedemos de forma análoga al caso básico anterior. Calculamos $c^d = (m^e)^d\ mod\ n = m$, consiguiendo el mensaje original de nuestro Amigo, *'m'*.

# 3.5 Ejemplo de cifrado/descifrado en bloque RSA

Vamos a mostrar cómo funciona el cifrado/descifrado en bloque para un criptosistema RSA. Para ello, primero procedemos a calcular los parámetros del sistema para después cifrar un mensaje de texto claro y decodificarlo.

Calculamos los valores para el módulo RSA, y los exponentes de codificación y decodificación. Elegimos dos números primos (en este ejemplo no vamos a tener en cuenta las consideraciones de seguridad de las claves RSA, para hacer el ejemplo más sencillo y comprensible) y calculamos el módulo RSA:

$$Sea\ p = 11\ y\ q = 23, entonces\ n = p * q = 253$$

Por lo tanto, $\varphi(n) = (p-1)(q-1) = 10 * 22 = 4 * 5 * 11$. Ahora necesitamos que el exponente de codificación cumpla (Ec. 3.1). El menor primo que la cumple es $e = 3$. Sólo nos queda obtener el exponente de decodificación. Para ello nos apoyamos en el Algoritmo Extendido de Euclides, obteniendo $d = 147$.

Una vez calculados los parámetros, pasamos a codificar un mensaje original. Supongamos que disponemos del siguiente alfabeto para nuestros mensajes: $\sum = \{a, b, c, d\}$.Le asignamos un valor a cada letra y calculamos la longitud del bloque de texto plano a codificar y la longitud del bloque con texto cifrado:

$$k = \lfloor \log_4 253 \rfloor = 3$$

| a | b | c | d |
|---|---|---|---|
| 0 | 1 | 2 | 3 |

Por lo tanto, la longitud del bloque con texto cifrado $es: k + 1 = 4$

Vamos a ver cómo ciframos el mensaje original “baba”. Según la tabla anterior, a este mensaje le corresponde el bloque “1010”. Este bloque está representado por el entero en base 4:

$$m = 1 * 4^3 + 0 * 4^2 + 1 * 4^1 + 0 * 4^0 = 64 + 4 = 68$$

Éste entero se codifica como:

$$c = m^e (mod\ n) = 68^3 (mod\ 253) = 206$$

Escribiendo *‘c’* en base cuatro tenemos:

$$c = 3 * 4^3 + 0 * 4^2 + 0 * 4^1 + 2 * 4^0 \rightarrow que\ se\ corresponde\ con \rightarrow 3002 \rightarrow \text{daab}$$

Luego nuestro texto cifrado en bloque es “daab”. Este mensaje está preparado para ser enviado a la persona con la que queramos contactar. Esta persona, una vez que lo recibe procede a decodificarlo:

Escribe el mensaje cifrado en base 4, recuperando el valor de $c = 206$. Aplicando la ecuación de decodificación (Ec. 3.2) tenemos:

$$c^d(mod\ n) = 206^{147}(mod\ n) = 68 = m$$

Que es nuestro mensaje original enviado.

# 3.6 Seguridad de RSA: el Problema de la Factorización

Por definición, factorizar un número entero positivo *'n'* significa encontrar dos números enteros positivos *'u'* y *'v'* tales que su producto sea igual a *'n'* y ambos números han de ser mayores que uno [INFA]. Tales números se conocen como *factores* (o *divisores*) de *'n',* y este se conoce denomina número compuesto. A la ecuación $n = u \times v$ se la denomina *factorización de 'n'*.

Aquellos números enteros positivos mayores que uno que no son compuestos, es decir, que no pueden ser factorizados, son conocidos como números *primos*. Diremos pues que una factorización de un número es prima si el número es resultado del producto de factores primos.

La factorización de un número entero se cree que es un problema difícil; a día de hoy esta afirmación no tiene un fundamento matemático en el que basarse. La única evidencia de que factorizar es un problema difícil reside en que no se ha podido encontrar hasta la fecha un algoritmo de factorización eficiente y rápido, pero no se descarta que en un futuro pueda ser encontrado.

Distinguimos dos clases de algoritmos para la factorización de un número:

- Algoritmos de propósito *específico* – Aquellos que trabajan de manera muy rápida gracias a la presencia de alguna propiedad especial que presentan ciertos números enteros. Su eficacia depende pues de que se den ciertas características en los factores del número a procesar.
- Algoritmos de propósito *general* – Aquellos que trabajan independientemente de las características de los factores del número, sólo dependen del tamaño del número a factorizar.

## 3.6.1 Algoritmos de propósito específico

A continuación vamos a describir brevemente algunos de los algoritmos de propósito específico más importantes, aunque en la actualidad se considera que ninguno de estos métodos es aplicable para factorizar números utilizados en criptosistemas reales. Sin embargo para otros números que puedan tener factores pequeños o presenten alguna característica peculiar, alguno de estos métodos puede sernos de gran ayuda. Para su

comprensión, asumiremos que *'n'*, que es el número a factorizar, es compuesto y no es potencia de un número primo.

#### 3.6.1.1. División por Tentativa

Es el algoritmo de factorización más simple e intuitivo. Se trata de buscar el factor primo más pequeño *'p'* de *'n'*, probando a dividir *'n'* por todos los números primos empezando por el dos hasta $\sqrt{n}$. Podemos apoyarnos en una tabla que ya tenga computados los $\sqrt{n}$ primeros números primos (calculada por ejemplo gracias a la Criba de Eratóstenes [POME]) para facilitar la búsqueda, aunque de esta forma perderíamos su cualidad principal: se trata de un algoritmo altamente paralelizable, como veremos más tarde. Asumiendo que tenemos dicha tabla, el proceso ejecuta $\varphi(p)$ divisiones, siendo *'$\varphi$'* la Función de Euler [BUCH, capítulo 2.17]. Sabemos que $\varphi(p) \approx p/\ln(p)$, lo que nos lleva a afirmar que encontrar el menor *'p'* de *'n'* cuesta alrededor de *'p'* pasos, lo cual es aceptable sólo para valores de *'n'* menor que $10^6$.

*EJEMPLO*: Sea $n = p \times q = 196.283 = 331 \times 593$ , donde *'p'* y *'q'* son números primos. Para factorizar *'n'* mediante división por tentativa, calculamos $l = \lceil \sqrt{n} \rceil$ , que va a ser el límite superior de números a probar.

- Si hemos obtenido una tabla de números primos mediante la Criba de Eratóstenes, vamos probando uno por uno cada primo de la tabla, de menor a mayor, hasta *'l'*, obteniendo así el menor factor primo de *'n'*, es decir, *'p'*. Si estamos trabajando con módulos RSA (como es el caso) para obtener el otro factor simplemente dividimos $n/p$. Si no, someteríamos al factor encontrado a un test de primalidad y si no se trata de un número primo, volveríamos a factorizarlo.

- Si no contamos con la tabla, simplemente empezaríamos a probar el primer número primo conocido, esto es, el 2. Si no es factor, los números pares los descartamos, así que seguiríamos con el siguiente, el 3 y de ahí en adelante, probando con todos los números impares hasta *'l'*. En nuestro ejemplo $l = \sqrt{n} = 444$, por lo que probaríamos hasta encontrar $p = 331$ y una vez obtenido este, conseguiríamos $q = {}^{n}/_{q} = 593$.

#### 3.6.1.2. Método $p-1$ de Pollard

Este método se basa principalmente en otro método del mismo autor, el Método Rho de Pollard [INFA, capítulo 3], simplemente reemplazando la paradoja del Cumpleaños por el Pequeño Teorema de Fermat [BUCH, capítulo 2.11]. Sea *'p'* un factor de *'n'*. Para cualquier entero *'a'* con $1 < a < p$ y $mcd(a,p) = 1$, según el Pequeño Teorema de Fermat tenemos que $a^{p-1} \equiv 1(mod\ p)$, por lo que $a^{k(p-1)} \equiv 1^k \equiv 1(mod\ p)$, para cualquier *'k'* entero. Por lo tanto, para cualquier múltiplo *'m'* de $(p-1)$ tendremos que $a^m \equiv 1(mod\ p)$, esto es lo mismo que decir que *'p'* divide a $a^m - 1$. Entonces si calculamos el $mcd(\ (a^m - 1)\ mod\ n, n\ )$ podemos obtener un factor de *'n'*.

Como candidatos a *'m'*, el Método $p-1$ usa el producto de todas las potencias primas menores que una cota $B$, o igualmente decir que $p-1$ es $B-liso$ con dicho límite $B$ (éste concepto lo veremos más adelante con más detalle). Por lo tanto, tenemos que *'m'*, con $m > 1$, se expresaría como $m = \prod_{q \in \mathbb{P}, q^e \leq B} q^e$. Si las potencias primas que dividen a $p-1$ son menores que $B$, entonces *'m'* es un múltiplo de $p-1$. El algoritmo

calcula $g = mcd(a^k - 1, n)$ para un *'a'* seleccionado adecuadamente. Si no se encuentra ningún divisor de *'n'* entonces se tendrá que elegir una nueva cota *'B'*.

*EJEMPLO*: Cojamos para nuestro ejemplo $n = 1.241.143, B = 13\ y\ a = 2$. Entonces obtenemos *'m'* como todas las potencias primas que son menores que la cota, esto es, 13. Tenemos que $m = 2^3 \times 3^2 \times 5^1 \times 7^1 \times 11^1 \times 13^1 = 8 \times 9 \times 5 \times 7 \times 11 \times 13$. Ahora calculamos el máximo común divisor de *'a'* elevado al *'m'* resultante y el número a factorizar, *'n'*, con un algoritmo eficiente como es el Algoritmo de Euclides: $mcd(2^m - 1, n) = 547$. Por lo tanto, $p = 547$ es un divisor de *'n'*. Dividiendo $n/p$ obtenemos el otro factor primo.

#### 3.6.1.3. Método de Curva Elíptica

Se trata del algoritmo más rápido dentro de los algoritmos de propósito especial. Este método fue sugerido por A. K. Lenstra en [INFA, capítulo 3] y posteriormente mejorado. Se trata de una generalización del Método $p - 1$ de Pollard: mientras que este sugiere trabajar en el grupo multiplicativo de un campo finito, Lenstra justifica su trabajo en el uso de grupos basados en los puntos de una curva elíptica. La teoría de curvas elípticas es una rama del álgebra bastante compleja que se escapa del ámbito de este texto, por lo que remitimos al lector interesado a [LUCE] y a [ELLE].

#### 3.6.1.4. Método de Fermat

Este método puede obtener fácilmente los factores de *'n'* si ambos están cercanos entre sí. Definimos el término 'cercanos' como que la distancia que hay entre dos números es pequeña, y el término 'distancia' como la resta del número menor al mayor. Sea $n = p \times q$, con $p, q\ primos\ impares$. Por lo tanto, siendo $p > q \rightarrow p - q = 2 \times d$, para una distancia *'d'* pequeña. Entonces $x = q + d \wedge y = d$, satisfacen $n = (x - y) \times (x + y)$, esto es,

$$n = x^2 - y^2$$

Para encontrar el valor de *'x'*, probamos calculando $x_1 = \lceil \sqrt{n} \rceil + 1, x_2 = \lceil \sqrt{n} \rceil + 1, \ldots$, hasta que $x^2 - n$ sea un cuadrado perfecto, para así tener $y^2 = x^2 - n$.

*EJEMPLO*: Supongamos para nuestro ejemplo que $n = 9017$. Calculamos $\lceil \sqrt{n} \rceil = x = 94$. Procedemos a calcular ${x_i}^2 - n$ buscando que el resultado sea también un cuadrado perfecto. Los cálculos se pueden ver en la siguiente tabla:

| $x_i + i$ | ${x_i}^2 - n$ |
|---|---|
| 95 | 8 |
| 96 | 199 |
| 97 | 392 |
| 98 | 587 |
| 99 | $784 = 28^2$ |

Tabla 3.1: cálculo de las potencias cuadradas sobre $n = 9017$ mediante el método de Fermat.

Por lo tanto tenemos que:

$$99^2 - n = 784 = 28^2 \text{ es un cuadrado perfecto, por consiguiente:}$$

$$x = 99 \wedge y = 28 \rightarrow 9017 = (99 - 28) \times (99 + 28) = \mathbf{71 \times 127}$$

Existen muchos más métodos de propósito especial para la factorización de números. Si el lector está interesado en ellos, véase [INFA] y [BUCH].

### 3.6.2 Algoritmos de propósito general

Los algoritmos de propósito general serán tratados en detalle en el capítulo 4.

## 3.7 Seguridad de las claves RSA

Como ya dijimos anteriormente, en la actualidad no se conoce un algoritmo eficiente capaz de factorizar un número lo suficientemente grande en tiempo polinómico. Si un enemigo (persona que quiere comprometer la seguridad de nuestro sistema) conociese los factores *'p'* y *'q'* que componen *'n'*, también conocería $\varphi(n) = (p-1) \times (q-1)$, y entonces sería capaz de conseguir *'d'* apoyándose en el exponente de codificación gracias al Algoritmo de Euclides Extendido. Concluimos que la seguridad de RSA radica en la dificultad de encontrar los factores primos que forman el módulo RSA.

- **Tamaño recomendado del módulo**.
  Según los Laboratorios RSA[25], el mejor tamaño para el módulo RSA depende en el tipo de seguridad que uno necesite para sus propósitos. Cuanto más grande sea la clave, mayor será la seguridad pero más lentas serán las operaciones del algoritmo RSA. Nos proponen elegir la longitud del módulo en base a dos consideraciones:
    - El valor de los datos protegidos y por cuánto tiempo necesitan ser protegidos.
    - La potencia de cálculo que uno posee.

  También recomiendan que la clave de cada usuario individual caduque cada cierto tiempo, ya que la clave que es útil hoy puede dejar de serlo mañana, gracias a los avances en potencia de cálculo.

  Actualmente los Laboratorios RSA recomiendan tamaños de clave de 1.024 bits para uso corporativo y de 2.048 para casos de mayor seguridad.

[25] http://www.rsa.com/rsalabs/node.asp?id=2218

- **Selección de los primos y de los exponentes.**

Para garantizar la seguridad de nuestro sistema conviene seguir una serie de recomendaciones en la configuración de los parámetros del algoritmo:

- El exponente de decodificación *'d'* ha de ser más grande que $n^{1/4}$. Para valores menores que dicha cota, existe un algoritmo en tiempo polinomial que calcula *'d'* mediante la expansión por fracciones continuas de $e/n$.
- Como ya hemos visto en la sección 3.5.1, existen algoritmos de factorización eficientes para factores primos que cumplen unas propiedades especiales. Es nuestra labor evitar estos factores:

  $|p - q|$ ha de ser grande; de esta forma evitamos posibles ataques mediante el método de potencias de cuadrados de Fermat (ver sección 3.5.4).

  Introducimos el concepto de *'primos fuertes'*. Un número primo es 'fuerte' si se satisfacen las siguientes condiciones: *'r'* es un factor primo grande de $p - 1$, $p + 1$ tiene un factor primo grande y $r - 1$ tiene un factor primo grande.

Los primos fuertes son necesarios para evitar los ataques de factorización $'p + 1'$ de Williams y $'p - 1'$ de Pollard (ver sección 5.3.2).

Sin embargo, aunque la elección de números primos fuertes evite este tipo de ataques, no resulta efectivo a la hora de resistir otro tipo de ataques más sofisticados, como puede ser la Criba en el Campo de Números (NFS), la cual será vista más adelante.

# 3.8 Ataques contra RSA: comprometiendo el entorno en el cual es usado

En esta sección cambiamos el enfoque a la hora de atacar el criptosistema RSA, centrándonos en describir ataques contra el entorno [SPRI] en el que este es usado en vez de criptoanalizar el algoritmo visto anteriormente en el punto 3.3.

Si el número de todos los posibles mensajes a cifrar es pequeño, es decir, tenemos un criptosistema en el cual el cardinal del conjunto de mensajes a cifrar *'M'* (ver sección 3.1) es pequeño, y además esos mensajes son conocidos de antemano, un atacante puede codificar todos los mensajes con la clave pública, que como ya sabemos, es conocida por todos. Entonces si intercepta un criptograma puede descifrarlo comparándolo con los que ya tiene calculados. Este método recibe el nombre de *Ataque a Espacio de Mensajes pequeño*.

Si tenemos una única clave RSA (pública-privada) y la utilizamos tanto para codificar mensajes como para firmar digitalmente estaremos comprometiendo la seguridad de nuestro criptosistema. La firma digital [BUCH, capítulo 11] garantiza que un mensaje recibido proviene de la persona que firma dicho documento. Asumamos que dicho criptosistema también se usa para autentificación mutua [BUCH, capítulo 13].

Ahora alguien, llamémosle Enemigo, nos pide que probemos nuestra identidad firmando un número aleatorio provisto por él. Dicho número ha sido generado con su clave privada. Entonces Enemigo verifica nuestra firma con nuestra clave pública, como parte del protocolo de firma digital. En este contexto, Enemigo puede atacar de forma satisfactoria nuestra clave con un simple cálculo. Supongamos que Enemigo intercepta un mensaje cifrado *'c'* que iba dirigido a nosotros. Entonces:

- Enemigo selecciona un $r \in \mathbb{Z}_n^*$ al azar. Ahora calcula $x = r^e \times c \ (mod\ n)$, donde $(n, e)$ es nuestra clave pública. Ahora nos envía $x$ para conseguir nuestra firma, esto es *'$x^d$'* , donde *'d'* es nuestra clave privada. Nótese que cuando recibimos *'x'*, nos parece que es un número totalmente aleatorio.
- Enemigo recibe el mensaje firmado por nosotros, y calcula $r^{-1}x^d$, esto es, el mensaje sin cifrar de *'c'*.

Este ataque es conocido como *Ataque sobre la Firma y Codificación de RSA*.

Se debe evitar que dos o más personas tengan el mismo módulo *'n'* en un criptosistema RSA. Cada usuario ha de tener un módulo distinto. Supongamos que sabemos que un cierto usuario Amigo posee el mismo módulo RSA que nosotros. Esto significa que conocemos los factores del módulo RSA de nuestro Amigo, ya que son los mismos que usamos nosotros (Teorema Fundamental de los Números Primos [BUCH, capítulo 1]), o podemos calcularlos a través de nuestros exponentes de cifrado y descifrado.

Ahora supongamos que una tercera persona, llamémosla Enemigo, tiene en su poder la información de que tanto nosotros como nuestro Amigo poseemos el mismo módulo RSA. Entonces Enemigo se encuentra con la posibilidad de comprometer tanto la seguridad de nuestras claves como las de nuestro Amigo, y por lo tanto, nuestros mensajes y los suyos. Sea $(n, e_1)$ nuestra clave pública y $(n, e_2)$ la clave de nuestro Amigo, y supongamos además que $mcd(e_1, e_2) = 1$, es decir, que son primos entre sí. Sea $m \in \mathbb{Z}_n$ un mensaje enviado tanto a nuestro Amigo como a nosotros, encriptado de la siguiente forma: $c_i = m^{e_i}, i = \{1, 2\}$. El problema reside en que el texto original puede ser calculado conociendo $c_i, e_1, e_2$ y $n$. Como $e_1\ es\ primo\ con\ e_2$, existen dos enteros *'r'* y *'s'* tales que $r \times e_1 + s \times e_2 = 1$, que pueden ser encontrados gracias al Algoritmo de Euclides Extendido. Uno de los dos números es negativo, por ejemplo *'r'*. Llegamos a la siguiente situación:

Si $c_1 \notin \mathbb{Z}_n^* \rightarrow$ Podemos factorizar *'n'* calculando $mcd(c_1, n)$, rompiendo así el criptosistema.

Si no, aplicamos de nuevo el Algoritmo de Euclides Extendido para calcular *'$c_1^{-1}$'*. Así podemos recuperar el mensaje '*m'* usando:

$$(c_1^{-1})^{-r} c_2^s = (m^{e_1})^r \times (m^{e_2})^s = m^{e_1 \times r + e_2 \times s} = m$$

Este ataque es denominado *Ataque a Módulo Común*.

Si los factores primos que componen el módulo RSA son elegidos aleatoriamente, entonces la probabilidad de que dos usuarios compartan el mismo *'n'* es despreciable.

En muchas aplicaciones prácticas, el proceso de codificación se realiza mediante algún dispositivo limitado, como por ejemplos las tarjetas inteligentes (tarjetas con un circuito integrado TCI, como pueden ser las tarjetas bancarias) [SMAR]. Debido a esa limitación, tener un exponente de codificación grande puede significar un alto coste en términos de energía de la batería. En un intento de simplificar y abaratar el consumo del proceso de encriptación un puede estar tentado en establecer dicho exponente a un número pequeño, por ejemplo, $e = 3$. Puede parecer a primera vista que el valor pequeño del exponente de codificación no afecta a la seguridad del sistema, ya que aún hay que intentar invertir $m^3(mod\ n)$ sin conocer la factorización de *'n'*, pero no es así. Un atacante enterado de dicha debilidad puede apoyarse en el Teorema Chino de los Restos, que dice lo siguiente:

*"Dadas e congruencias* $x \equiv a_i\ (mod\ m_i)$ *tales que cada* $m_1, \dots, m_e$ *son primos entre sí dos a dos, existe un único* $x\ (mod\ m_1 \times m_2 \times \dots \times m_e)$ *que satisface todas las congruencias, y este puede ser calculado de forma eficiente."*

Podemos verlo con un ejemplo. Supongamos que enviamos el mismo mensaje a tres personas diferentes, llamémoslas Antonio, José y Alberto. Codificamos el mensaje con cada una de las claves públicas pertenecientes a cada destinatario, generando los siguientes textos cifrados:

$$c_{AN} = m^3(mod\ n_{AN}), c_{JO} = m^3\big(mod\ n_{JO}\big), c_{AL} = m^3(mod\ n_{AL})$$

Ahora cualquiera que intercepte, llamémosle J.Luis, los tres mensajes cifrados puede recuperar fácilmente el mensaje original *'m'*. Sin pérdida de generalidad podemos asumir que los módulos RSA de cada destinatario son primos entre sí (ya que si no es así, J.Luis tendría la capacidad de encontrar un factor no trivial de una de las claves públicas simplemente calculando el máximo común divisor y podría decodificar el mensaje original directamente). Aplicando el Teorema Chino de los Restos, J.Luis calcula un número *'c'* tal que $c = m^3\ (mod\ n_{AN} \times n_{JO} \times n_{AL})$. Podemos ver que como $m < n_{AN} \wedge m < n_{JO} \wedge m < n_{AL} \rightarrow\ m^3 < n_{AN} \times n_{JO} \times n_{AL}$ y por lo tanto $c = m^3$ sobre los enteros. Entonces el mensaje original puede ser recuperado simplemente calculando la raíz cúbica de *'c'* sobre los enteros. Este ataque recibe el nombre de *Ataque al Exponente de Codificación Pequeño*.

A primera vista podemos pensar que este problema puede ser evitado si nunca enviamos el mismo mensaje a varias personas, pero esto no es así, como podemos comprobar en [TELA].

Como en el anterior caso, puede ser deseable elegir un exponente de decodificación pequeño *'d'*, de forma que la decodificación se haga lo más eficiente posible. De nuevo se plantea la misma problemática que para el exponente de codificación. De esta forma estaríamos hablando de un *Ataque al Exponente de decodificación Pequeño.* Para evitar ataques de este estilo, es conveniente que el exponente de encriptación sea de un tamaño aproximado a *'n'*.

# 4. Métodos de propósito genérico de factorización

## 4.1 Introducción

En el capítulo anterior hemos visto algunos de los principales métodos de factorización de números enteros. En esta sección vamos a ampliar algunos de ellos, los cuales sientan las bases y los fundamentos principales del método de factorización más rápido conocido en la actualidad, denominado Criba General en el Campo de Números (General Number Field Sieve ~ GNFS).

## 4.2 Congruencia de cuadrados

### 4.2.1 Introducción

En el apartado 3.5.4 vimos el método de Fermat para la factorización un número utilizando la técnica de la diferencia de cuadrados para números cercanos entre sí. En la década de 1920, Maurice Kraitchik [POME] propuso una mejora del método de factorización de Fermat. La idea que propuso fue:

*"Si en vez de intentar encontrar dos números enteros cuya diferencia de cuadrados sea el número a factorizar, esto es,* $p, q \in \mathbb{Z}^+ \; tales \; que \; p^2 - q^2 = n$, *es suficiente encontrar tales p, q cuya diferencia de cuadrados sea múltiplo de n, esto es,*

$$p^2 \equiv q^2 (mod\ n)\text{''} \; (Ec. 4.1)$$

Éste método es conocido como congruencia de cuadrados, en la cual se buscan números enteros *'x'* e *'y'* tales que

$$x^2 \equiv y^2 (mod\ n)$$

Pero $x \not\equiv y\ (mod\ n)$. Entonces, si $n = p \times q$ tenemos:

$$x^2 \equiv y^2 (mod\ n) \Leftrightarrow n\ | x^2 - y^2 \Leftrightarrow n | (x + y) \times (x - y) \Leftrightarrow$$

$$\Leftrightarrow p \times q | (x + y) \times (x - y) \Leftrightarrow p\ | (x + y) \text{ ó } p\ | (x - y) \Leftrightarrow q\ | (x + y) \text{ ó } q\ | (x - y).$$

Esto significa que podemos encontrar *'p'* y/o *'q'* calculando $mcd(x \pm y, n)$, donde $mcd(a, b)$ denota el máximo común divisor de dos números *'a'* y *'b'*, el cual puede ser eficientemente calculado haciendo uso del Algoritmo de Euclides [BUCH, capítulo 1.8].

Pero no siempre encontramos con este método un factor no trivial, como podemos comprobar en la siguiente tabla:

| $p \mid (x+y)$ | $p \mid (x-y)$ | $q \mid (x+y)$ | $q \mid (x+y)$ | $mcd(x+y,n)$ | $mcd(x-y,n)$ | ¿factor? |
|---|---|---|---|---|---|---|
| *Sí* | *Sí* | *Sí* | *Sí* | *n* | *n* | *No* |
| *Sí* | *Sí* | *Sí* | *No* | *n* | *p* | ***Sí*** |
| *Sí* | *Sí* | *No* | *Sí* | *p* | *n* | ***Sí*** |
| *Sí* | *No* | *Sí* | *Sí* | *n* | *q* | ***Sí*** |
| *Sí* | *No* | *Sí* | *No* | *n* | *1* | *No* |
| *Sí* | *No* | *No* | *Sí* | *p* | *q* | ***Sí*** |
| *No* | *Sí* | *Sí* | *Sí* | *q* | *n* | ***Sí*** |
| *No* | *Sí* | *Sí* | *No* | *q* | *p* | ***Sí*** |
| *No* | *Sí* | *No* | *Sí* | *1* | *n* | *No* |

Tabla 4.1: Diferentes resultados de la congruencia de cuadrados con $n = p \times q$.

Si asumimos que las combinaciones de la tabla son equiprobables tenemos una probabilidad de $^2/_3$ de obtener un factor no trivial de *'n'*.

Además, Kraitchik introduce una forma de "jugar con los resultados" obtenidos en la búsqueda de cuadrados perfectos para reducir el tiempo de esta. Para ello, introduce el uso de la función $f(x)$, definida como $f(x) = x^2 - n$. Se trata de intentar encontrar varios números *'x'* cuyo producto de sus correspondientes $f(x)$ sea igual a un cuadrado. Entonces

$$Si\ f(x_1) \times \ldots \times f(x_k) = q^2\ \wedge\ x_1 \times \ldots \times x_k = p\ entonces$$

$$p^2 = {x_1}^2 \times \ldots \times {x_k}^2 \equiv (x_k^2 - n) \times \ldots \times (x_k^2 - n) = f(x_1) \times \ldots \times f(x_k) = q^2 (mod\ n) \qquad (Ec. 4.2)$$

Por lo que hemos encontrado una solución a la Ec. 4.1. Para resolverlo, sólo necesitamos una forma de calcular el conjunto $x_1 \times \ldots \times x_k$ . Para esto, Kraitchik se dio cuenta de que algunos de los resultados de la función $f(x)$ se factorizaban en potencias de primos pequeños, esto es, su factorización en producto de primos se podía calcular fácilmente y, fijándose en los exponentes de las potencias de los primos que los factorizaban, se dio cuenta de que el producto de algunos de ellos resultaba en un cuadrado perfecto. De este modo se había encontrado una forma para obtener el conjunto $x_1 \times \ldots \times x_k$ y resolver la Ec. 4.2.

### 4.2.2 Ejemplo

Veamos este resultado con un ejemplo. Supongamos que queremos factorizar $n = 2041$. Siguiendo los pasos del Método de Fermat, calculamos $x = \lceil \sqrt{n} \rceil + 1 = 46$. A partir de este número, obtenemos los resultados de la función $f(x)$ para la secuencia, recogidos en la siguiente tabla:

| $x_i + i$ | $f(x_i + i) = (x_i + i)^2 - n$ |
|---|---|
| 46 | 75 |
| 47 | 168 |
| 48 | 263 |
| 49 | 360 |
| 50 | 459 |
| 51 | 560 |

Tabla 4.2: Resultados de la función $f(x)$ para el ejemplo.

Por ahora vemos que no hemos obtenido ningún cuadrado perfecto, por lo que en este punto el Método de Fermat seguiría buscando valores posteriores a 51. Sin embargo, Kraitchik indaga en la factorización de los resultados de la función $f(x)$ para ver si con la combinación de algunos de ellos consigue formar un cuadrado. Por lo que tendríamos:

| $x_i + i$ | $f(x_i + i) = (x_i + i)^2 - n$ | Factorización | Elegida |
|---|---|---|---|
| 46 | 75 | $75 = 3 \times 5^2$ | ⇐ |
| 47 | 168 | $168 = 2^3 \times 3 \times 7$ | ⇐ |
| 48 | 263 | $263 = 263$ | |
| 49 | 360 | $360 = 2^3 \times 3^2 \times 5$ | ⇐ |
| 50 | 459 | $459 = 3^3 \times 17$ | |
| 51 | 560 | $560 \equiv 2^4 \times 5 \times 7$ | ⇐ |

Tabla 4.3: Elección adecuada de resultados en el ejemplo $n = 2041$.

Si nos fijamos, el producto de las factorizaciones elegidas de la tabla… ¡forma un cuadrado!

$$75 \times 168 \times 360 \times 560 = 2^{10} \times 3^4 \times 5^4 \times 7^2 = (2^5 \times 3^2 \times 5^2 \times 7)^2$$

Por lo tanto, ya tenemos una solución a la Ec. 4.1 con la forma:

$$p = 46 \times 47 \times 49 \times 51 \equiv 311 \ (mod\ 2041)$$

$$q = 2^5 \times 3^2 \times 5^2 \times 7 \equiv 1416 \ (mod\ 2041)$$

Como $311 \not\equiv 1416 \ (mod\ 2041)$, podemos usar el Algoritmo de Euclides para calcular el máximo común divisor $mcd(1416 - 311, 2041) = 13$, y tener pues la factorización del número que buscábamos, esto es, $2041 = 13 \times 157$.

# 4.3 Método de fracciones continuas (CFRAC)

Más tarde, en 1931, D.H. Lehmer y R. E. Powers sugirieron cambiar la función de Kraitchik $Q(x) = x^2 - n$ por otra derivada de la expansión de fracciones continuas (CFRAC) de $\sqrt{n}$. Este método utiliza una manera diferente a la hora de buscar cuadrados. Una fracción continua es una representación numérica utilizada para representar números reales con números enteros. Cualquier número real *'x'* se puede representar con una fracción continua general

$$x = a_0 + \cfrac{b_1}{a_1 + \cfrac{b_2}{a_2 + \cfrac{b_3}{a_3 + \cdots}}}$$

Con $a_i,\ b_i \in \mathbb{Z}$. Su aplicación a la búsqueda de cuadrados perfectos es la siguiente:

Si $a_i/b_i$ es la i-ésima fracción continua convergente a $\sqrt{n}$, sea $Q_i \equiv a_i^2 - b_i^2 \times n$. Entonces $Q_i \equiv a_i^2 \ (mod\ n)$. Por lo que ahora en vez de manejar los resultados de la función $Q(x)$ de Kraitchik, podemos "jugar" con los números *'$Q_i$'* para calcular cuadrados, ya que en ambos casos son congruentes módulo *'n'*.

La razón por la cual Lehmer y Powers sugieren este método reside en la desigualdad $|Q_i| < 2 \times \sqrt{n}$. Los números *'$Q_i$'* son más pequeños (en valor absoluto) que los números obtenidos con $Q(x)$. Es mucho más fácil y cómodo manejar números pequeños a la hora de intentar hallar productos para calcular cuadrados. Es por esta razón por la que se prefiere el método de estos dos matemáticos frente al polinomio cuadrático de Kraitchik.

# 4.4 Las mejoras de Morrison y Brillhart

En 1975, Michael A. Morrison y John Brillhart dieron un paso adelante en la investigación de factorización de números enteros. Definieron una estrategia para la búsqueda de una subsecuencia a partir de una secuencia con el producto de cuadrados basada en la noción de álgebra lineal de vector de exponentes. En [MOBR] los autores aplican el método de fracciones continuas y la estrategia del vector de exponentes para la factorización del séptimo número de Fermat $F_7$.

Todo número entero positivo *'m'* tiene asociado un vector de exponentes $v(m)$ basado en la factorización en números primos de *'m'*. Más concretamente, sea $p_i$ el i-ésimo factor primo de *'m'*, entonces $m = \prod {p_i}^{v_i}$ es la factorización en números primos de *'m'*. Nótese que dicha factorización es sobre el producto de los infinitos números primos, pero solo algunos primos tendrán como exponente $v_i \neq 0$. Entonces el $v(m)$ es el vector $(v_1, v_2, \dots, v_k)$ .

Para la factorización de enteros mediante congruencia de cuadrados sólo nos interesa obtener cuadrados, esto es, que todas las componentes del vector de exponentes sean números pares. Para ello podemos reducir los elementos del vector módulo 2. Al estar manejando exponentes, la multiplicación de factores se convierte en suma de exponentes, por lo que necesitamos buscar números tales que la suma de sus vectores de exponentes resulte el vector cero (módulo 2). Para ilustrar esto, podemos ver que si tomamos como referencia el ejemplo anterior:

$$v(75) \equiv (0,1,0,0)\ mod\ 2$$

$$v(168) \equiv (1,1,0,1)\ mod\ 2$$

$$v(360) \equiv (1,0,1,0)\ mod\ 2$$

$$v(560) \equiv (0,0,1,1)\ mod\ 2$$

Sumando los cuatro vectores, obtenemos el vector cero (módulo 2), esto implica que $75 \times 168 \times 360 \times 560$ es un cuadrado. Por lo tanto Morrison y Brillhart para sistematizar el procedimiento proponen un método que está dividido en dos etapas:

- Recolección de datos útiles.
- Procesamiento de dichos datos.

En la primera parte, coleccionamos números que mantienen la propiedad de que 'factorizan fácilmente' (que ya citamos anteriormente en el Método de Kraitchik). Vamos a definir formalmente dicha propiedad y, además, el concepto de *Base de factores:*

- Se dice que un número '*m*' es B-liso si todos los factores primos de *'m'* son menores o iguales a *B*.
- Una *base de factores* es un conjunto no vacío $\mathcal{F}$ de números enteros primos positivos. Se dice pues que un número entero *'k' es liso sobre la base de factores* $\mathcal{F}$ si todos los factores primos de *'k'* son miembros de $\mathcal{F}$.

Morrison y Brillhart entonces sugieren que se fije un número *B* y que sólo escojamos aquellos números de la secuencia que factorizan completamente sobre los primeros *B* primos, es decir, que son *B*-lisos sobre la base de factores $F$. Cuando se han recogido al menos $B + 1$ números que cumplen dicha propiedad, tendremos a nuestra disposición $B + 1$ vectores de exponentes en el espacio vectorial *B-dimensional* $\mathbb{F}_2^B$, donde $\mathbb{F}_2$ denota el Campo Finito de Galois $GF(2)$. Gracias al álgebra lineal sabemos que pueden haber vectores que sean linealmente dependientes, es decir, que al menos uno de ellos se puede expresar como combinación lineal de los demás, lo que en $\mathbb{F}_2$ significa simplemente que la suma de un subconjunto de elementos sea el vector cero. Para ello contamos con diversos algoritmos que nos permiten encontrar estas dependencias y que serán explicados más adelante.

A la hora de establecer la base de factores, Morrison y Brillhart proponen prescindir de aquellos números primos *'p'* para los cuales el número a factorizar *'n'* no es residuo cuadrático módulo *'p'*, ya que nunca dividirán ni a $Q_i$(CFRAC) ni a $f(x)$ (Kraitchik). Además, algunos de los números que nos interesen en la fase de recolección pueden ser

negativos, por lo que añadiremos '-1' a la base de factores para tratar con estos también. Para representar este cambio, añadimos en el vector de exponentes una coordenada más, que sea '0' para los números positivos y '1' para los negativos.

# 4.5 El algoritmo de Dixon como ejemplo de las mejoras de Morrison y Brillhart

## 4.5.1 Introducción

Ahora, para ilustrar todos los conceptos mencionados del método de Morrison y Brillhart proponemos el estudio del Algoritmo de Dixon, el cual es menos eficiente que CFRAC pero muestra de manera precisa y sencilla todos los conceptos anteriormente estudiados.

El *Algoritmo de Dixon* es una versión de la aproximación de Morrison y Brillhart para el problema de la congruencia de cuadrados [INFA], en el cual se muestran los conceptos de *base de factores* y de *'ser liso' sobre una base de factores*, conceptos clave para el posterior estudio de métodos de factorización más sofisticados como la Criba Cuadrática y de GNFS.

Comenzamos fijando la base de factores $\mathcal{F} = \{p_1, p_2, \ldots, p_m\}$. Para ello escogemos de manera arbitraria una cota superior '$B$'. La base de factores va a estar formada por todos los números primos menores o iguales que dicha cota, además del '-1'. Buscamos, a partir de $\lceil\sqrt{n}\rceil$ (como ya vimos anteriormente en el método de Fermat), un conjunto de enteros de la forma $x_i = i + \lceil\sqrt{n}\rceil, i \in \mathbb{Z}^+$que cumplen la siguiente propiedad:

$$f(x_i) = x_i^2 (mod\ n)\ es\ liso\ sobre\ \mathcal{F}$$

Una vez encontrados *'m'* enteros con dicha propiedad, podemos encontrar un subconjunto $\mathcal{U}$ de enteros en la secuencia tales que:

$$\prod_{x_i \in \mathcal{U}} f(x_i) = p_1^{2e_1} \times p_2^{2e_2} \times \ldots \times p_m^{2e_m} = (p_1^{e_1} \times p_2^{e_2} \times \ldots \times p_m^{e_m})^2$$

Con $e_i \geq 0$. Entonces si establecemos que:

$$x = \prod_{x_i \in \mathcal{U}} x_i \ \ e\ y = p_1^{e_1} \times p_2^{e_2} \times \ldots \times p_m^{e_m}$$

Y tenemos la diferencia de cuadrados construida como:

$$x^2 = \prod_{x_i \in \mathcal{U}} r_i \equiv \prod_{x_i \in \mathcal{U}} f(x_i) \equiv y^2 (mod\ n)$$

Hemos utilizado el subconjunto $\mathcal{U}$ para producir diferencias de cuadrados. Encontrar este subconjunto es una tarea fácil, como veremos más adelante.

### 4.5.2 Ejemplo

Nos centramos ahora en ver cómo funciona el Algoritmo de Dixon con un ejemplo sencillo y práctico:

Sea $n = 143 = 13 \times 11$. Utilizamos este número cuya factorización es bien conocida por sencillez. Elegimos una cota $B = 5$, esto quiere decir que la factorización de cualquier número que elijamos ha de estar formada sólo por primos pertenecientes a $\mathcal{F} = \{2, 3, 5\}$. Ésta será nuestra base de factores (para mayor sencillez y mejor comprensión, hemos escogido un ejemplo sin números negativos, por lo que prescindimos en la base de factores de '-1'). Ahora calcularíamos $\lceil \sqrt{n} \rceil = 12$ y procederíamos según la fórmula $x_i = i + \lceil \sqrt{n} \rceil, i \in \mathbb{Z}^+$. Para una mejor comprensión del algoritmo, elijamos $a = 17$, ya que como veremos más tarde, ninguno de los números en el intervalo [12,16] nos interesa. Sea entonces este número $a = 17$. Tenemos que

$$a_1^2 = 289 = 3 + 2 \times 143 \equiv 3 \; mod \; 143$$

Por lo que $f(a) = 3 \; es \; liso \; sobre \; \mathcal{F}$. Guardamos el par $\big(a, f(a)\big) = (17, 3)$. A la identidad $17^2 \equiv 2^0 \times 3^1 \times 5^0 \; (mod \; 143)$ la denominaremos *relación*. Necesitamos encontrar *'m'* relaciones, siendo *'m'* el cardinal de la base de factores $\mathcal{F}$, esto es, 3. Volvemos a ejecutar la misma operación para el siguiente número así como lo hacíamos con el Método de Fermat, esto es, para $a = 18$. Para facilitar los cálculos, podemos darnos cuenta de que:

$$(a + 1)^2 = a^2 + 2 \times a + 1$$

Teniendo esto en cuenta, procedemos:

$$a_2^2 = 18^2 = 17^2 + 2 \times 17 + 1 \equiv 3 + 34 + 1 \; (mod \; 143) \equiv 2 \times 19 (mod \; 143)$$

Pero vemos que $a_2^2 \; no \; es \; liso \; sobre \; \mathcal{F}$, ya que 18 no pertenece a la base de factores, luego no nos sirve y lo descartamos. Seguimos buscando más enteros de la misma forma, obteniendo los siguientes resultados:

| Nº Aleatorio $a_i$' | $f(a_i) = a^2\ (mod\ 143)$ | Relación | ¿ $\mathcal{F}$ − Smooth? |
|---|---|---|---|
| 12 | 1 | $12^2 \equiv 2^0 \times 3^0 \times 5^0\ (mod\ 143)$ | No |
| 13 | 26 | $13^2 \equiv 2^1 \times 13\ (mod\ 143)$ | No |
| 14 | 53 | $14^2 \equiv 53\ (mod\ 143)$ | No |
| 15 | 82 | $15^2 \equiv 2^1 \times 41\ (mod\ 143)$ | No |
| 16 | 113 | $16^2 \equiv 113\ (mod\ 143)$ | No |
| 17 | 3 | $17^2 \equiv 2^0 \times 3^1 \times 5^0\ (mod\ 143)$ | **Sí** |
| 18 | 38 | $18^2 \equiv 2^1 \times 19\ (mod\ 143)$ | No |
| 19 | 75 | $19^2 \equiv 2^0 \times 3^1 \times 5^2\ (mod\ 143)$ | **Sí** |
| 20 | 114 | $20^2 \equiv 2^1 \times 3^1 \times 19\ (mod\ 143)$ | No |
| 21 | 12 | $21^2 \equiv 2^2 \times 3^1 \times 5^0\ (mod\ 143)$ | **Sí** |

Tabla 4.4: Resultados del ejemplo para el Algoritmo de Dixon.

En la tabla hemos empezado a calcular para $a = \lceil \sqrt{n} \rceil = 12$. Como ya dijimos antes, escogimos el valor inicial de $a = 17$ para facilitar la comprensión del ejemplo y entender por qué no nos sirven los números del '12' al '16'.

Ahora tenemos que combinar las relaciones obtenidas, de tal forma que consigamos congruencias de cuadrados:

$$(17 \times 19)^2 \equiv 2^0 \times 3^2 \times 5^2 (mod\ 143) \equiv (3 \times 5)^2 (mod\ 143) \qquad (Ec. 4.3)$$

$$(19 \times 21)^2 \equiv 2^2 \times 3^2 \times 5^2 (mod\ 143) \equiv (2 \times 3 \times 5)^2 (mod\ 143) \qquad (Ec. 4.4)$$

$$(17 \times 21)^2 \equiv 2^2 \times 3^2 \times 5^0 (mod\ 143) \equiv (2 \times 3)^2 (mod\ 143) \qquad (Ec. 4.5)$$

De Ec. 4.3 obtenemos:

$$x = 17 \times 19, y = 3 \times 5 \rightarrow mcd(323 - 15, 143) = 11\ y\ mcd(323 + 15, 143) = 13$$

Es decir, los factores primos *'p'* y *'q'* que buscábamos. De Ec. 4.4 obtenemos:

$$x = 19 \times 21, y = 2 \times 3 \times 5 \rightarrow mcd(399 - 30, 143) = 1\ y\ mcd(323 + 15, 143) = 143$$

Que son factores triviales de $n = 143$. Finalmente de Ec. 4.5 obtenemos:

$$x = 17 \times 21, y = 2 \times 3 \rightarrow mcd(357 - 6, 143) = 13\ y\ mcd(357 + 6, 143) = 11$$

Que son también los factores buscados. Este proceso de selección de relaciones y combinación de ellas entre sí lo hemos hecho a mano y de forma trivial, comprobando cuadrados, ya que se trata de un ejemplo muy simple. Vamos a detallar ahora el método sugerido por Morrison y Brillhart para encontrar la mejor combinación de relaciones:

Supongamos que disponemos de un conjunto $\mathcal{V}$ de relaciones obtenidas en la fase de recolección. En el siguiente paso, el procesamiento de lo recolectado, tenemos que

elegir un subconjunto $\mathcal{W} \subset \mathcal{V}$ de tal forma que cuando multipliquemos los elementos de $\mathcal{W}$ entre sí resulte una solución de la congruencia $x^2 \equiv y^2 \ (mod\ n)$. Para ello:

A) Para cualquier $\mathcal{W} \subset \mathcal{V}$, el producto de los 'lados izquierdos' de las relaciones $\prod_{v \in \mathcal{W}} v^2$ es un cuadrado, ya que se trata de un producto de cuadrados.
B) Sin embargo, el producto de las correspondientes 'partes derechas' de las relaciones no siempre forman un cuadrado:
   a. Para cada primo *'p'* perteneciente a la base de factores, el exponente que le corresponde en el producto sobre $\mathcal{W}$ es la suma de los exponentes de *'p'* en las relaciones de $\mathcal{W}$. Por ejemplo:

$$17^2 \equiv 2^0 \times 3^1 \times 5^0 (mod\ 143)$$
$$19^2 \equiv 2^0 \times 3^1 \times 5^2 (mod\ 143)$$
$$(17 \times 19)^2 = 17^2 \times 19^2 \equiv (2^0 \times 3^1 \times 5^0) \times (2^0 \times 3^1 \times 5^2)(mod\ 143) \equiv$$
$$\equiv (2^{0+0} \times 3^{1+1} \times 5^{0+2})(mod\ 143) \equiv$$
$$\equiv (3^2 \times 5^2)\ (mod\ 143) \equiv (3 \times 5)^2 (mod\ 143)$$

   Los exponentes de los elementos de la base de factores para el producto sobre $\mathcal{W}$ vienen dados por el vector suma de los vectores de las relaciones de $\mathcal{W}$. Para nuestro ejemplo:

$$Vector17 = (0, 1, 0);\ Vector19 = (0, 1, 2)$$
$$VectorSuma = (0 + 0, 1 + 1, 0 + 2) = (0, 2, 2)$$

   b. Esta suma de exponentes no siempre es par, lo que implica que las partes derechas no siempre forman un cuadrado, como se dijo anteriormente. Por lo tanto para encontrar $\mathcal{W}$ tal que el producto de las 'partes derechas' sea también un cuadrado necesitamos buscar un subconjunto de vectores cuya suma sea un vector con todas las entradas par. Como ya vimos anteriormente, según Morrison y Brillhart sólo hace falta reducir cada componente del vector módulo dos y esperar que la suma de vectores sea el vector nulo.

Llegados a este punto, podemos reducir el problema de encontrar 'lados derechos' que son también cuadrados al problema de encontrar todas las combinaciones pares de vectores. Éste es un problema clásico de álgebra lineal, para el que existen distintos algoritmos de resolución, como el método de Gauss clásico, el método de bloques de Lanczos o el de Wiedemann, algunos de los cuales serán vistos más adelante. Todos ellos, en general, siguen las mismas pautas: hay *'m'* relaciones y *'k'* números primos que forman la base de factores. Por lo tanto tenemos una matriz de $m \times k$, donde cada *'m'* corresponde al vector formado por las *k-tuplas* de los exponentes en las relaciones de *'m'*. Con el ejemplo anteriormente visto tendríamos:

$$\begin{pmatrix} 0 & 1 & 0 \\ 0 & 1 & 2 \\ 2 & 1 & 0 \end{pmatrix} = \begin{pmatrix} Vector17 \\ Vector19 \\ Vector21 \end{pmatrix} = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \end{pmatrix} (mod\ 2)$$

Si $m > k$ , existen al menos $m - k$ combinaciones pares de filas (es decir, de vectores de dimensión *k* cuya suma es par), cada una de las cuales nos conduce a una

solución independiente para factorizar *'n'*. Concluimos que si conseguimos suficientes relaciones obtendremos siempre la factorización del número.

### 4.5.3 Tiempo de ejecución

Según [KIMI], el número de operaciones estimado para encontrar un factor no trivial de un módulo RSA está acotado por la siguiente fórmula:

$$e^{(2+o(1))\sqrt{\log n \log\log n}}$$

## 4.6 Criba Cuadrática (QS): Mejorando lo presente

### 4.6.1 Introducción

La Criba Cuadrática (QS) fue inventada en 1981 por Carl Pomerance, extendiendo las ideas de Kraitchik y de Morrison y Brillhart. QS ha sido el método de factorización más rápido conocido hasta la llegada de la Criba en el Campo de Números ("Number Field Sieve"), allá por el año 1993. Aun así, hoy en día es considerado el método más rápido cuando se trata de factorizar números de hasta 110 dígitos.

La Criba Cuadrática se basa en la idea de un método tan antiguo como simple: la Criba de Eratóstenes. Ésta es un procedimiento para obtener números primos hasta una cierta cota, en el cual se empieza desde el número primo más bajo, esto es, el dos, y se van señalando los múltiplos de dos como números compuestos. Dos es entonces etiquetado como primo, y se sigue con el siguiente número no marcado, que es primo, el tres. Pomerance se dio cuenta de que la Criba de Eratóstenes hacía algo más que encontrar primos: algunos números eran señalados más de una vez, lo que significaba que tenían más de un factor primo. Esto significa que podría explorar rápidamente el intervalo en busca de aquellos números que han sido marcados más de una vez, lo que implica encontrar números con muchos factores primos pequeños.

Para su aplicación en la Criba Cuadrática, en vez de marcar aquellos que tienen varios factores primos, pensó que si un número primo *'p'* divide al número a comprobar *'n'*, se podría hacer la división de dicho número a lo largo del intervalo hasta obtener '1'. Entonces si al final de la criba *'n'* se ha convertido en '1', significa que factoriza completamente sobre ese intervalo. Si el intervalo lo acotamos con *'B'*, significa que el número es *B-liso*.

Tenemos que observar que esta forma, hasta ahora, no considera potencias de primos, por lo que por ejemplo el número '20' terminaría el '2', ya que no se ha

considerado que $20 = 2^2 \times 5$. Para tener esto en cuenta, se divide el número *'n'* entre la mayor potencia del primo posible de forma que ahora si se obtendría la factorización completa.

De esta forma podemos reconocer muy rápidamente qué números son *B*-lisos en un intervalo. Lo que hace que la criba funcione de forma eficiente es que para cada módulo *'p'* en la criba, los múltiplos del número *'p'* aparecen en sitios regulares en el intervalo, como veremos más adelante.

### 4.6.2 Conceptos teóricos

La Criba Cuadrática es, esencialmente, el algoritmo de Dixon mejorado. En esta ocasión, pasamos de tener un algoritmo probabilista (Dixon) a un algoritmo completamente determinista. Los conceptos de base de factores, 'ser liso' sobre una base de factores y dependencias sobre vectores de exponentes vistos en él siguen vigentes en QS. La gran mejora sobre el método de Dixon radica en la modificación del polinomio de búsqueda $f(x_i)$ y en el cambio de la política de búsqueda de elementos *'$x_i$'* como ya hemos visto anteriormente:

- Pasamos de $f(x_i) = {x_i}^2 \ (mod\ n)$, donde $x_i = i + \lceil \sqrt{n} \rceil$, a $f(x_i) = {x_i}^2 - n$.

Es decir, recuperamos el polinomio que sugirió Kraitchik.

- Vamos a apoyarnos en la siguiente propiedad para la búsqueda de elementos *'$x_i$'*:

$$Si\ p | f(x_i), entonces\ p \mid f(x_i + k \times p), para\ todo\ k \in \mathbb{Z}.$$

La demostración es sencilla. Fijamos *'$x_i$'* y $p\ primo \in \mathbb{Z}^+$. Supongamos que $p \mid f(x_i)$, entonces

$$f(x_i) \equiv 0\ (mod\ p) \overset{def}{\Longrightarrow} {x_i}^2 - n \equiv 0\ (mod\ p). Entonces\ para\ todo\ k \in \mathbb{Z}$$

$$f(x_i + k \times p) = {x_i}^2 + (2 \times x_i \times k \times p) + (k^2 \times p^2) - n \equiv {x_i}^2 - n \equiv 0\ (mod\ p)$$

Gracias a esta propiedad se puede abordar de un modo distinto el problema de la búsqueda de elementos *'$x_i$'* cuyo $f(x_i)$ sea liso sobre una base de factores. En vez concentrarnos en un $f(x_i)$ fijo y tratar de averiguar qué números primos lo dividen y por tanto comprobar si es liso sobre la base de factores, hecho costoso ya que implica probar mediante división por tentativa [véase apartado 3.5.1] para cada elemento perteneciente a la base de factores, escogemos un primo $p \in F$, donde *F* es la base de factores. Entonces determinamos qué valores de $f(x_i)$ son divisibles por ese primo *'p'* fijado, dentro de un rango establecido para *'$x_i$'*. De esta forma podemos ahorrarnos el tiempo de tratar dividir entre los primos que no dividen a $f(x_i)$. Recordemos que la división es una de las operaciones básicas más costosas computacionalmente hablando.

El trabajo para calcular valores de *'$x_i$'* tales que $f(x_i)$ es divisible por *'p'* se reduce a resolver la congruencia cuadrática ${x_i}^2 \equiv n\ (mod\ p)$, que puede ser resuelta de manera rápida y eficiente. Gracias a la propiedad anteriormente citada, el resto de valores para los cuales $f(x_i)$ es divisible por *'p'* se calculan como $x_i + k \times p, para\ k \in \mathbb{Z}$. Para que

este procedimiento funcione, el número *'n'* ha de ser un residuo cuadrático módulo *'p'*, por lo tanto no debería de haber en $F$ ningún número primo para el cual *'n'* no sea un residuo cuadrático.

### 4.6.3 Estructura del algoritmo

Una vez vista la base teórica, pasamos al algoritmo en la práctica y posteriormente veremos un ejemplo práctico:

*1) Configuración de la base de factores y del intervalo de criba.*

Nos interesa que $f(x_i)$ sea lo más pequeño posible. De esta forma aumentamos la probabilidad de que 'sea liso' sobre la base de factores que vamos a establecer. Para ello, debemos elegir $i \in r_i$ lo más cercano a cero, por lo que elegimos una cota '$M$' y sólo consideramos valores de *'i'* tales que $-M \leq i \leq M$. A este intervalo le denominaremos 'Intervalo de Criba' (o 'IdeC'). Entonces para $i \in [-M, M]$, *'p'* primo tal que $p \mid f(r_i)$ tenemos:

$$(i - \lceil \sqrt{n} \rceil)^2 \equiv n \ (mod\ p)$$

*'n'* es un residuo cuadrático módulo *p*, y por lo tanto los primos contenidos en la base de factores deben ser primos tales que el Símbolo de Legendre [PACE]

$$\left(\frac{n}{p}\right) = 1$$

Además, la base de factores contendrá también el elemento '-1' y estará limitada por una cota '*B'*, siendo todo número que factorice sobre dicha base de factores *B-liso*.

En la elección de las cotas 'M' y 'B', Eric Landquist en [LAND] sugiere las siguientes fórmulas para la optimización del rendimiento:

$$B = \left(e^{\sqrt{\ln(n)\ln(\ln(n))}}\right)^{\sqrt{2}/4}$$
$$M = \left(e^{\sqrt{\ln(n)\ln(\ln(n))}}\right)^{3\times\sqrt{2}/4}$$

*2) Proceso de criba.*

La QS introduce una nueva manera para buscar números y comprobar si factorizan sobre nuestra base de factores, obteniendo una mejora sustancial del rendimiento sobre el Algoritmo de Dixon. La principal diferencia es que en vez de para cada número del intervalo a tratar, comprobar mediante división por tentativa si es *B-liso*, vamos a trabajar con el intervalo completo de una vez, gracias a la propiedad que ya hemos visto:

$$Si\ p\ es\ un\ factor\ primo\ de\ f(x_i), entonces\ p \mid f(x_i + k \times p), k \in \mathbb{Z}. \quad (Ec.\ 4.6)$$

Procedemos como sigue:

- Para cada factor primo de la base de factores, resolver la siguiente congruencia, que es básicamente el cálculo de los ceros de la función $f(x_i)$:

$$f(x_i) = \left(x_i + \lfloor\sqrt{n}\rfloor\right)^2 - n \equiv 0\ (mod p) \rightarrow (x_i + \lfloor\sqrt{n}\rfloor)^2 \equiv n\ (mod\ p)$$

Si llamamos $s = x_i + \lfloor\sqrt{n}\rfloor$ tenemos:

a. Si $p = 2\ (caso\ particular)$:
$s^2 \equiv n\ (mod\ 2) \rightarrow s = 1, entonces\ s_p = 1 - \lfloor\sqrt{n}\rfloor (mod\ 2)$

b. Si $p > 2$:
$s^2 \equiv n\ (mod\ p) \rightarrow Dos\ soluciones: x_1, x_2 = p - x_1. Entonces$
$s_{p1} = x_1 - \lfloor\sqrt{n}\rfloor (mod\ p)\ y\ s_{p2} = x_2 - \lfloor\sqrt{n}\rfloor\ (mod\ p)$ $(Ec. 4.7)$

- A partir de este punto, podemos actuar de dos formas distintas:

A) Versión clásica:

Inicializar el Intervalo de Criba como un vector cuyos elementos sean los valores de $f(x_i)$.

Para cada solución obtenida en el punto anterior, llamémosla '$s_p$', dividir $f(s_p)$ por *'p'*. Gracias a Ec. 4.6 sabemos que todos los elementos del intervalo que pertenecen a la progresión aritmética $s_p \pm k \times p$ también son divisibles por *'p'*, por lo que nos ahorramos todas las divisiones por esos elementos. Así continuamos para todos los primos de la base de factores, con el objetivo de que, al final del proceso obtengamos unos en varias posiciones del vector. Estas posiciones representan los números que factorizan completamente sobre nuestra base de factores, es decir, los que nos interesan.

Hemos de tener en cuenta las potencias de primos, como ya vimos anteriormente, por lo que en vez de dividir directamente por el primo, calculamos la mayor potencia de dicho primo que divide a $f(s_p)$.

B) Podemos implementar también otra versión del proceso de criba, menos exacta pero mucho más rápida [LAND]:

En vez de trabajar con los valores de $f(x_i)$ sobre el intervalo, vamos a inicializar un vector a cero.

Ahora, para cada elemento del intervalo que cumple la propiedad Ec. 4.6, le sumamos el número de bits del factor con el que se cumple la progresión aritmética. Cada posición del vector va a contener a lo largo del proceso la suma del número de bits de cada factor de la base de factores que divide a $f(x_i)$.

Después de haber procesado todos los elementos de la base de factores de esta forma, aquellas posiciones del vector cuyo valor sea próximo al valor de $(\log f(x_i))$, el cual es el número de bits de $f(x_i)$, son probables de factorizar completamente sobre nuestra

base de factores. Sólo nos queda comprobar con división por tentativa que realmente lo son. Con este método se consigue mejorar el tiempo de ejecución de la versión clásica.

> NOTA: También se podría haber hecho a la inversa: almacenar el número de bits de cada $f(x_i)$ en un vector $(\log f(x_i))$ y en todas aquellas posiciones que cumplan la propiedad [1] restar el número de bits del factor. Aquellas posiciones con valores próximos a cero son las candidatas a factorizar sobre la base de factores.

Hay que recalcar que no hemos tenido en cuenta con este método las potencias de primos. Para evitar volver a cribar el subintervalo de candidatos obtenidos en la primera pasada, introducimos el concepto de condición umbral, por lo que todo elemento factorizable sobre la base de factores deberá cumplir dicha condición. Esta condición, de acuerdo a la fórmula de Robert D. Silverman [SILV]:

$$\left[\log\left(\frac{M\sqrt{k \times {}^{n}/_{2}}}{{p_{max}}^{T}}\right)\right] \approx \frac{1}{2}\ln(n) + \ln(M) - T\ln(p_{max})$$

Donde:

$$M = cota\ del\ Intervalo\ de\ Criba$$
$$p_{max} = mayor\ primo\ de\ la\ Base\ de\ Factores$$
$$T = valor\ alrededor\ de\ 2\ y\ p_{max}$$

En el siguiente proyecto[26]se sugiere, basándose en los resultados obtenidos por Silverman, la siguiente regresión sobre *'T':*

$$T = 0.02688x + 0.783929$$

Donde $x = nº\ digitos\ de\ n$. Por lo que en el apartado c), comprobaremos mediante división por tentativa aquellas posiciones cuyo valor sea mayor que *'T'*.

3) *Construcción y reducción de la matriz*.

El proceso iterativo de criba se realiza hasta obtener al menos $B + 1$ elementos del intervalo de criba que factorizan completamente sobre la base de factores. Ahora colocamos todos los vectores de exponentes módulo dos (elementos de $GF(2)$) formando una matriz 'A', de forma que las columnas de 'A' representan los exponentes módulo dos de los primos de la base de factores. Como queremos que el producto de cada $f(x_i)$ sea un cuadrado perfecto, necesitamos que la suma de los exponentes de cada factor primo en la base de factores sea par, esto es, el vector nulo (módulo dos).

Por lo tanto, necesitamos encontrar tantos elementos como números primos hay en la base de factores, esto es, *'B'*. Como vimos antes en la tabla 4.1, como mínimo la mitad de las relaciones del espacio de soluciones nos dará un factor primo no trivial. Por lo tanto, si encontramos $B + 10$ elementos que factorizan sobre nuestra base de factores, encontraremos un factor propio del número con una probabilidad de $\frac{1023}{1024}$.

[26]http://www.bytopia.dk/qs/ Contacto: michael@bytopia.dk

Para reducir la matriz y hallar un conjunto de soluciones al sistema se pueden utilizar distintos métodos, como puede ser el método de bloques de Wiedemann [WIED], el método de bloques de Lanczos [BRIG capítulo 4.4] o algunas versiones que mejoran el método de resolución de Gauss, como por ejemplo el método de los cuatro rusos [RUSS].

4) *Solución*.

Sólo nos queda comprobar los vectores solución obtenidos en el anterior apartado para ver si su correspondiente producto ($f(x_i)$ y $x_i$) nos entrega un factor propio del número a factorizar. Para ello, y siguiendo las bases de la 'Congruencia de Cuadrados' ya explicadas, calculamos el Máximo Común Divisor y obtenemos dicho factor. Si resulta que en vez de ello, obtenemos un factor trivial, comprobamos el siguiente elemento del conjunto que abarca el espacio de soluciones y repetimos la operación.

Cuando un factor propio es encontrado, obtenemos el otro factor con una simple división y los sometemos a un test de primalidad [FUCR, capítulo 9.4], para comprobar que realmente hemos terminado la factorización del número. Como estamos considerando módulos RSA, es decir, números compuestos por el producto de dos números primos, sabemos de antemano que los factores no triviales obtenidos son primos, por lo que hemos terminado nuestro trabajo.

Una de las grandes ventajas de la Criba Cuadrática sobre otros métodos de factorización es la facilidad para distribuir las tareas entre distintos ordenadores. Cuando se propone paralelizar un algoritmo, se estudia de forma que la parte más costosa del algoritmo es la que se intenta distribuir en varias tareas de forma que el tiempo de ejecución de esa parte se vea reducido drásticamente. Como ya hemos visto, la Criba Cuadrática se descompone en cuatro pasos principales. Se puede comprobar fácilmente que el paso más costoso, con diferencia, es la parte referente al proceso de criba, por lo que nos planteamos su paralelización.

Más tarde veremos una variante de la Criba Cuadrática clásica altamente paralelizable, la Criba Cuadrática Multi-Polinomial, basada en la construcción de distintos polinomios para que cada procesador trabaje sobre un único polinomio y de esta forma obtener números que factorizan completamente sobre la base de factores, pero trabajando sobre un intervalo de criba mucho más pequeño.

### 4.6.4 Ejemplo

Vamos a representar todas las ideas anteriormente explicadas mediante un ejemplo práctico:

Sea $n = 87463$ un módulo RSA, esto es, un número compuesto producto de dos números primos. Los pasos del algoritmo de Criba Cuadrática son los siguientes:

1. *Configuración de la base de factores y del intervalo de criba.*
   Calculamos los parámetros del algoritmo para el *'n'* dado:

$$B = \left[\left(e^{\sqrt{\ln(n)\ln(\ln(n))}}\right)^{\sqrt{2}/4}\right] = 6$$

$$M = \left[\left(e^{\sqrt{\ln(n)\ln(\ln(n))}}\right)^{3\times\sqrt{2}/4}\right] = 264$$

Por lo que el rango del Intervalo de Criba es $[-M, M] = [-264, 264]$. Para buscar los integrantes de nuestra Base de Factores, comprobamos que cada número primo sea residuo cuadrático módulo *'n'*, desde el tres en adelante hasta que obtengamos $B$ elementos ('2' y '-1' siempre van a ser integrantes de la Base de Factores por convenio, contado el primero para el cálculo del número de valores de la 'BdeF'):

| p | 3 | 5 | 7 | 11 | 13 | 17 | 19 | 23 | 29 |
|---|---|---|---|---|---|---|---|---|---|
| $\left(\frac{n}{p}\right)$ | 1 | -1 | -1 | -1 | 1 | 1 | 1 | -1 | 1 |

Tabla 4.5. Resultados del número de Legendre para los $\varphi(30)$ primeros primos.

Quedando entonces como: $BdeF = [-1, 2, 3, 13, 17, 19, 29]$. El Intervalo de Criba se inicializa a cada valor de $f(x_i)$ según la ecuación del polinomio.

2. *Proceso de criba*.

Resolvemos la ecuación [2] para buscar cada par de soluciones (excepto para '2'):

| p | 2 | 3 | 13 | 17 | 19 | 29 |
|---|---|---|---|---|---|---|
| x | 1 | 1, 2 | 5, 8 | 7, 10 | 5, 14 | 12, 17 |
| sp1, sp2 | 0 | 0, 1 | 9, 12 | 1, 4 | 4, 14 | 7, 12 |

Tabla 4.6. Soluciones a la ecuación Ec. 4.7 para la Base de Factores.

Los valores de *'sp1'* y *'sp2'* son los que nos interesan, ya que van a actuar de 'raíz' en la progresión aritmética para calcular todos los elementos del Intervalo de Criba que son divisibles por cada factor de la Base de Factores.

De esta forma, por ejemplo, para $p = 2$ tenemos que todos los elementos en posiciones pares del Intervalo de Criba son divisibles por *'p'*. La siguiente tabla muestra, a modo de ejemplo, el proceso de criba para los elementos del subintervalo comprendido entre [0, 12]: $(\text{Nota: } \sqrt{n} = 295)$.

| $x_i$ | $x_i + \sqrt{n}$ | $f(x_i)$ | 2 | 3 | 13 | 17 | 19 | 29 | -1 | $f(x_i)$final | ¿B-liso? |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 295 | -438 | X | X | | | | | X | 73 | No |
| **1** | 296 | 153 | | X | | X | | | | 1 | Sí |
| **2** | 297 | 746 | X | | | | | | | 373 | No |
| **3** | 298 | 1341 | | X | | | | | | 149 | No |
| **4** | 299 | 1938 | X | X | | X | X | | | 1 | Sí |
| **5** | 300 | 2537 | | | | | | | | 2537 | No |
| **6** | 301 | 3138 | X | X | | | | | | 523 | No |
| **7** | 302 | 3741 | | X | | | | X | | 43 | No |
| **8** | 303 | 4346 | X | | | | | | | 2173 | No |
| **9** | 304 | 4953 | | X | X | | | | | 127 | No |
| **10** | 305 | 5562 | X | X | | | | | | 103 | No |
| **11** | 306 | 6173 | | | | | | | | 6173 | No |
| **10** | 307 | 6786 | X | X | X | | | X | | 1 | Sí |

Tabla 4.7. Proceso de criba para el subintervalo [0, 12]. (La tabla no muestra las potencias de primos que dividen a $f(x_i)$).

En este subintervalo habríamos encontrado tres elementos que factorizan completamente sobre nuestra Base de Factores. En la pasada hemos ido almacenando las potencias de los exponentes que los dividen. El vector de exponentes lo transformamos en un vector de exponentes módulo 2. Los datos están recogidos en la siguiente tabla:

| $x_i + \sqrt{n}$ | Factorización | Vector de exponentes | Vector de exponentes (módulo 2) |
|---|---|---|---|
| 296 | $3^2 \times 17$ | $[0, 2, 0, 1, 0, 0, 0]$ | $[0, 0, 0, 1, 0, 0, 0]$ |
| 299 | $2 \times 3 \times 17 \times 19$ | $[1, 1, 0, 1, 1, 0, 0]$ | $[1,1, 0, 1, 1, 0, 0]$ |
| 307 | $2 \times 3^2 \times 13 \times 29$ | $[1,2, 1, 0, 0, 1, 0]$ | $[1, 0, 1, 0, 0, 1, 0]$ |

Tabla 4.8. Elementos que factorizan sobre la Base de Factores para el ejemplo anterior.

3. *Construcción y reducción de la matriz.*

Una vez terminado el proceso de criba, tenemos al menos $B + 1$ elementos. Si no es así, necesitaríamos volver a ejecutar el paso anterior, pero haciendo más grande el Intervalo de Criba.

Construimos la matriz con los vectores de los exponentes y la reducimos mediante alguno de los métodos algebraicos disponibles (Wiedemann, Lanczos, Gauss, etc.). Tendremos como resultado un conjunto de posibles soluciones, por lo que vamos probando cada una de ellas en el siguiente paso. Para nuestro ejemplo, tenemos los siguientes vectores de exponentes después del proceso de criba (nos basta con siete elementos, ya que la base de factores está compuesta de seis números primos), que colocamos en una matriz de la siguiente forma (ya reducida módulo 2):

$$\begin{pmatrix} 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 \end{pmatrix} * \bar{v} = \bar{0}$$

Una posible solución es $\bar{v} = (1, 1, 1, 0, 1, 0)$, tomando las cuatro primeras columnas. Con sus valores $x_i$ asociados tenemos:

$$x = 265 * 278 * 296 * 307 = 6694540240 \equiv 34757\ (mod\ n).$$
$$y = \sqrt{(265^2 - n) * (278^2 - n) * (296^2 - n) * (307^2 - n)} =$$
$$= 2 * 3^4 * 13^2 * 17 * 29 = 13497354 \equiv 28052\ (mod\ n)$$

*4. Solución.*
Sólo falta calcular el máximo común divisor de los resultados obtenidos en el apartado anterior:

$$mcd(x - y, n) = 149, mcd(x + y, n) = 587$$

Si en este proceso no obtenemos factores no triviales de *'n'*, buscaríamos otra solución del apartado anterior y repetiríamos el proceso.

Como conocemos de antemano que el número a factorizar es un módulo RSA, sabemos que está compuesto por el producto de dos primos, por lo que si encontramos factores no triviales hemos terminado el proceso. Si no estuviéramos trabajando con un módulo RSA, tendríamos que comprobar con algún test de primalidad si el factor obtenido es primo. Si no es así, continuaríamos el proceso de factorización con el número encontrado.

### 4.6.5 Tiempo de ejecución

La Criba Cuadrática es un algoritmo de factorización determinista con una complejidad conjeturada en [SMPO]

$$e^{(1+o(1))\sqrt{\log n \log\log n}}$$

Lo que hace que sea el algoritmo elegido cuando se trata de factorizar números 'duros' (es decir, números que no presentan ciertas propiedades que pueden ser explotadas por otros algoritmos de factorización de propósito específico) desde 20 a 110 dígitos. A partir de los 110 dígitos, GNFS obtiene mejores rendimientos, aunque la frontera de los 110 dígitos no está bien definida, ya que la aparición de nuevas arquitecturas computacionales puede introducir variaciones y mejoras en los métodos en los que se basan.

Existen numerosas variaciones del algoritmo que consiguen mejoras sobre la Criba Cuadrática, pero aún así su complejidad asintótica sigue siendo representada por dicha fórmula. La variante más importante de la QS es la Criba Cuadrática Multi-Polinomial, que será vista a continuación.

### 4.6.6 MPQS – Criba Cuadrática Multi-Polinomial – Variante de la QS

Esta variante de la Criba Cuadrática fue sugerida por Peter Montgomery y como su propio nombre indica, utiliza varios polinomios de grado dos en vez de $f(x_i)$, todos de la forma

$$f(x_i) = ax^2 + 2bx + c$$

La idea de esta variante es que al utilizar varios polinomios, podemos trabajar sobre un intervalo de criba mucho más pequeño que en la Criba Cuadrática clásica, haciendo que $f(x_i)$ sea mucho más pequeño y por tanto aumentamos la probabilidad de que estos valores factoricen completamente sobre nuestra base de factores.

Para la elección de los coeficientes de cada polinomio, procedemos como:

Sea '*a*' un cuadrado. Elegimos *'b'* con las siguientes restricciones

$$0 \leq b < a\,, \qquad b^2 \equiv n\ (mod\ a)$$

Esto sólo se cumple si '$n$' es un cuadrado módulo '$q$', donde '$q$' representa cada factor que divide a *'a'*. Por los que nos es conveniente elegir un coeficiente *'a'* con una factorización conocida tal que *'n'* cumpla que es residuo cuadrático módulo *'q'*, esto es,

$$\left(\frac{n}{q}\right) = 1, donde\ q \mid a$$

Ahora sólo nos queda definir *'c'*, según la siguiente ecuación

$$b^2 - 4ac = n$$

La comprobación de que esta elección de coeficientes es la correcta es la siguiente:

$$a \times f(x_i) = (ax_i)^2 + abx_i + ac = (ax + b)^2 - n \rightarrow$$
$$(ax + b)^2 \equiv a \times f(x_i)\ (mod\ n)$$

Como *'a'* es un cuadrado, $f(x_i)$ también lo es.

Esta variante de Criba Cuadrática fue utilizada para factorizar el reto que propusieron los laboratorios RSA, el RSA-129. Cuando este fue anunciado, en 1977, pensaban que factorizar ese número costaría 23.000 años. Los investigadores, comandados por Lenstra y Manasse, que utilizaron MPQS tardaron ocho meses en descifrar su mensaje "The magic words aresqueamish ossifrage" (Las palabras mágicas son quebrantahuesos aprensivos).

# 4.7 GNFS: la barrera de los 110 dígitos

## 4.7.1 Introducción

Como ya hemos mencionado anteriormente, la Criba General del Cuerpo de Números es el método de factorización de números enteros grandes más rápido conocido, sobrepasando la barrera de los 110 dígitos en la que la Criba Cuadrática se limita. Este hecho lo convierte en el mejor algoritmo para atacar claves del criptosistema RSA.

GNFS es la generalización de SNFS y, ambos derivan de un algoritmo inicial conocido como NFS ("Number Field Sieve", o Criba en el Cuerpo de Números) [TNFS]. El algoritmo fue propuesto en 1988 por J.M. Pollard. [DEVE]. Se trata de un método para factorizar números enteros de la forma

$$r^e - s, donde\ 'r'\ y\ |s| son\ positivos, r > 1\ y\ 'e' es\ grande \qquad (Ec. 4.8)$$

El algoritmo combina técnicas tradicionales de factorización heredadas de la Criba Cuadrática con nuevas ideas en cuerpos de números algebraicos. La idea principal es trabajar en anillos algebraicos distintos de $\mathbb{Z}$, ya que estos anillos potencialmente pueden tener impuesta una noción de 'ser liso' similar a la de $\mathbb{Z}$, con la esperanza de encontrar muchos más valores lisos que en $\mathbb{Z}$. Una vez encontrados, si descubrimos una función que establezca una relación entre tales anillos y $\mathbb{Z}/_{n\mathbb{Z}}$ entonces podríamos hallar una forma de producir diferencias de cuadrados.

La base de este algoritmo inicial evolucionó a lo que se conoce como "Special Number Field Sieve" (SNFS), válido única y exclusivamente para números de la forma vista en Ec. 4.8. Más tarde se pudo eliminar las restricciones de la SNFS, desembocando en lo que se conoce como GNFS.

En la actualidad, algunas versiones de GNFS han conseguido factorizar claves RSA de 512 bits, y el siguiente paso, las claves de 1024 están siendo objetivo de futuras factorizaciones, no muy lejanas. Como ya vimos en el apartado 3.6 del capítulo anterior, la empresa de seguridad Laboratorios RSA sigue confiando en las claves de 1024 para uso corporativo, pero si es necesario extremar la seguridad de ciertos datos, nos recomiendan ya usar tamaños de clave de 2048. Esto quiere decir que no en mucho tiempo, ya sean nuevas mejoras en las implementaciones de GNFS o con nuevos

algoritmos de factorización, se podrá comprometer la seguridad de las claves de uso corporativo.

GNFS, al igual que la Criba Cuadrática, es altamente paralelizable, basándose en los principios de búsqueda de elementos válidos sobre un intervalo de criba.

### 4.7.2 Generalización de la Criba Cuadrática

Para poder trabajar con otros anillos distintos de $\mathbb{Z}$ y $\mathbb{Z}/_{n\mathbb{Z}}$, generalizamos la función que juega el polinomio $f(x_i)$ en la Criba Cuadrática. Como vimos en la QS,

$$f(x_i) = x_i - n$$

Lo que puede ser visto como un homomorfismo de anillos: $f: \mathbb{Z} \rightarrow \mathbb{Z}/_{n\mathbb{Z}}$, *'f'* mapea un cuadrado en el anillo $\mathbb{Z}$ en un cuadrado en el anillo $\mathbb{Z}/_{n\mathbb{Z}}$, con lo que obtenemos los enteros *'x'* e *'y'*. Si suponemos que existe un anillo R, con la noción de 'liso', y un homomorfismo $\phi$ entre dicho anillo y $\mathbb{Z}/_{n\mathbb{Z}}$ tendríamos:

$$Si\ a \in R\ con\ \phi(a^2) = \mathrm{y}^2 (mod\ n) y\ x = \phi(\mathrm{a}) \text{entonces}$$
$$x^2 \equiv \phi(\mathrm{a})^2 \equiv \phi(\mathrm{a}^2) \equiv \mathrm{y}^2 (\mathrm{mod\ n})$$

Con lo que hemos conseguido una diferencia de cuadrados. Si somos capaces de encontrar un elemento en R que sea cuadrado perfecto en R y en $\mathbb{Z}/_{n\mathbb{Z}}$, entonces gracias al homomorfismo $\phi$ obtendremos la diferencia de cuadrados buscada.

→ **Nota**: las nociones algebraicas que se ven a continuación son complejas. Si alguno de los conceptos no se conoce, remitimos al lector a los siguientes textos: [SHOU] y [IVOR].

### 4.7.3 El algoritmo

El algoritmo se divide en cinco fases:

1. *Selección del polinomio*

La elección de un polinomio que se comporte bien, un "buen polinomio", se encuentra en fase de investigación, ya que aún no se tiene claro dicho concepto. Se intenta que un "buen polinomio" sea aquel que consiga muchos valores lisos en un intervalo de criba. Dos son las características principales que hacen que un polinomio sea bueno:

- Tamaño – Los valores obtenidos por $f(x_i)$ son pequeños.

- Propiedades de la raíz – Si $f(x_i)$ tiene muchas raíces módulo primos pequeños.

Además, el polinomio tiene que cumplir las siguientes propiedades:

- Es irreducible sobre $\mathbb{Z}[x]$.
- Tiene una raíz módulo *'n'*.

Para una lectura más profunda sobre este tema, remitimos al lector a [MURP].

En la práctica, se utiliza un truco: expresar el número a factorizar como un polinomio con coeficientes pequeños, esto es:

$$n = a_d m^d + a_{d-1} m^{d-1} + \cdots a_0$$

Para ello necesitamos configurar dos parámetros:

1. El polinomio $f(x)$: $\mathbb{R} \rightarrow \mathbb{R}$ con coeficientes enteros.
2. Un entero $m \in \mathbb{Z}$ tal que se satisfaga la ecuación: $f(m) \equiv 0\ (mod\ n)$, es decir, una raíz del polinomio.

Después de elegir el grado del polinomio según la Tabla 4.9, escogemos el valor de *'m'* como $m = \sqrt[d]{n}$, aproximadamente.

| Dígitos (n) | <50 | 50-80 | 50-80 | <110 |
|---|---|---|---|---|
| Grado (polinomio) | 2 | 3 | 4 | 5 |

Tabla 4.9: Grado del polinomio según el número de dígitos del número a factorizar.

*2. Bases de factores.*

Este es el dominio sobre el que el algoritmo trabajará. En la Criba Cuadrática, la Base de Factores es compuesta por todo aquel primo *'p'* que cumpla que el número a factorizar *'n'* es residuo cuadrático módulo *'p'*. Ahora, como trabajamos en otros anillos, necesitamos introducir tres Bases de Factores con distintos objetivos:

- *Base de Factores Racional* – Consiste en buscar números primos en $\mathbb{Z}$, tal y como se hacía para la QS.
- *Base de Factores Algebraica* – La idea es elegir un conjunto de ideales primos $\mathcal{I}$ del anillo con el que trabajamos ($\mathbb{Z}[\theta]$, donde $\theta$ es la raíz del polinomio) y encontrar pares $(r, p)$ para los cuales el elemento $r + p\theta$ tiene un ideal principal $\langle r + p\theta \rangle$ el cual factoriza completamente en ideales primos de $\mathcal{I}$. Tales elementos son lisos sobre la Base de Factores Algebraica. Siguiendo la idea análoga a la QS, recolectando más parejas que ideales en $\mathcal{I}$ podemos esperar que alguno de los valores $r + p\theta$ correspondientes a esas parejas puede ser multiplicado para producir un cuadrado perfecto en $\mathbb{Z}[\theta]$.
  - Los pares $(r, p)$ cumplen $f(r) \equiv 0\ (mod\ p)$, con *'p'* perteneciente a la BFR. Buscar ideales primos se reduce a buscar los dichos pares.
- *Base de Caracteres Cuadráticos* – Es utilizada para determinar si el producto de los ideales pertenecientes a la BFA es un cuadrado perfecto en $\mathbb{Z}[\theta]$.

3. *Criba.*

Es el cuello de botella del algoritmo ya que se itera sobre un dominio muy grande de elementos. Para reducir tiempo de procesamiento, se puede trabajar con logaritmos, como ya vimos en la QS. En esta fase buscamos parejas de elementos $(a, b)$ tales que:

- $mcd(a, b) = 1$
- $a + bm$ factoriza completamente sobre BFR y $a + b\theta$ factoriza completamente sobre BFA.

4. *Álgebra lineal.*

Una vez finalizado el proceso de criba, tendremos una lista de pares $(a, b)$ los cuales son lisos sobre BFR y sobre BFA. Nuestro objetivo en esta fase es encontrar una combinación de elementos de dicha lista, como ya hicimos en la Criba Cuadrática, cuyo producto sea un cuadrado. Para ello, seguiremos la idea de Morrison y Brillhart de los vectores de exponentes y su reducción módulo dos, buscando el vector nulo (ver apartado 4.4). Para la obtención de una solución, se construye la matriz sobre $GF(2)$) y se procesa mediante algún método de reducción de matrices (Wiedemann, Lanczos, Gauss, etc.).

5. *Raíz cuadrada.*

Del paso anterior obtendremos una o más soluciones. Finalmente comprobamos si el resultado al que nos conduce cada una de las soluciones encontradas nos ofrece un factor no trivial del número a factorizar. Para ello, procedemos de la misma manera que vimos en QS, calculando el máximo común divisor de la suma y la resta de cada resultado obtenido con el número a factorizar.

En la siguiente imagen [Figura 4.1] se pueden ver de forma esquematizada todos los pasos de GNFS [CHIN].

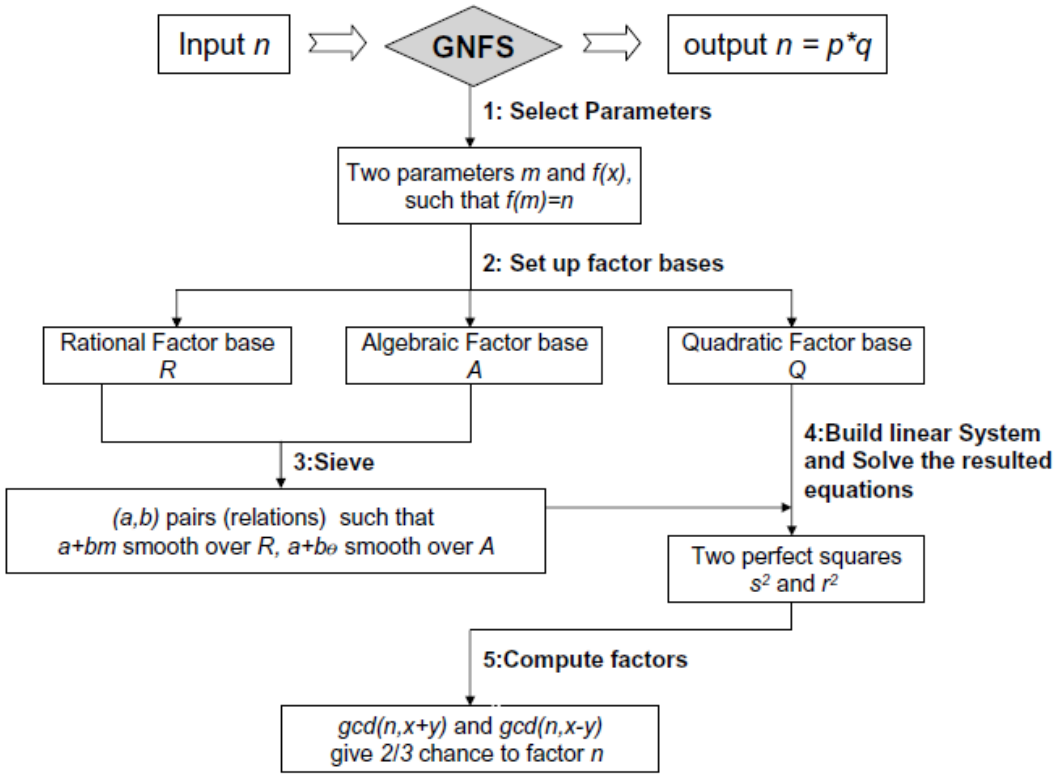


Figura4.1: Diagrama de las fases del algoritmo GNFS.

### 4.7.4 Tiempo de ejecución

GNFS dispone de una complejidad asintótica estimada en:

$$e^{(c+o(1))(\sqrt[3]{\log n})(\sqrt[3]{(\log\log n)^2})}$$

Donde $c = \sqrt[3]{64/9}$

# 5. Sistema RSA@Cloud

## 5.1 Introducción

RSA es el criptosistema de clave pública más extendido en la actualidad, habiendo resistido al ataque continuo de criptoanalistas desde su publicación. Transacciones bancarias, compras a través de la red, tramitaciones burocráticas… son sólo unos pocos ejemplos de la extensión y la importancia de dicho criptosistema. Debido al papel que juega en la sociedad, la invulnerabilidad de las claves RSA se convierte en un objetivo prioritario para cualquier ciudadano, más aún para gobiernos y empresas.

Para la evaluación de la seguridad de una clave que pueda utilizar una persona u organismo se necesitan herramientas que nos indiquen su resistencia a posibles ataques por parte de otros individuos u organismos. La herramienta Forecaster nos ofrece una estimación aproximada de la garantía de una clave en términos de tiempo, coste y tipo de máquina con la que se intenta comprometer su validez. Una vez realizado dicho estudio, la herramienta Engine pone a prueba su seguridad llevando a cabo un ataque de factorización al módulo RSA distribuido en una red de máquinas, ya sea a través del Cloud público de Amazon o de una red propia. Por último, la herramienta Codeswarm nos ofrece una representación visual de la distribución de las tareas entre los distintos servidores remotos y la máquina cliente.

El objetivo del sistema presentado en este documento reside en sentar las bases que permitan la ejecución paralela de cualquier implementación de un algoritmo de factorización de números en la Nube. También se pretende demostrar el potencial de la computación en nube para desarrollar trabajos de investigación científica [JUVE] o de ámbito empresarial en un tiempo óptimo, sin la necesidad de invertir sumas importantes en la instalación de una infraestructura física de computación.

A continuación se detallan las características arquitectónicas y funcionales del conjunto de herramientas que forman el sistema RSA@Cloud.

# 5.2 Desarrollo del proyecto: plan de trabajo

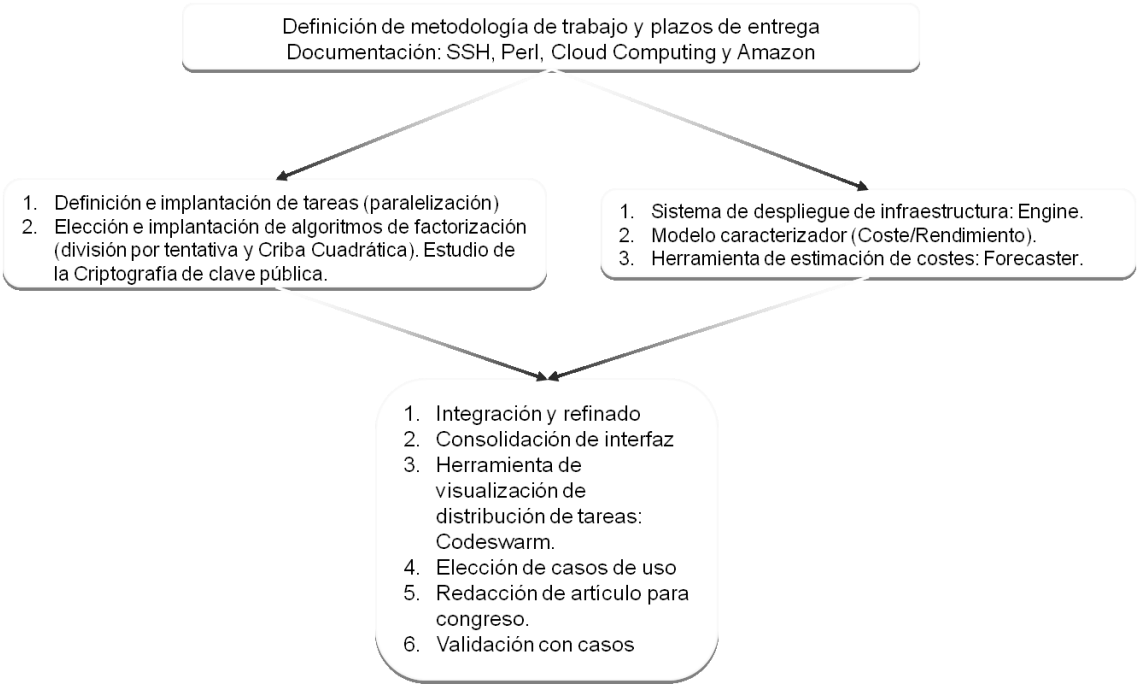


Figura 5.1: Plan de trabajo del proyecto RSA@Cloud.

# 5.3 Descripción de la Arquitectura

## 5.3.1 Engine

### 5.3.1.1. Introducción.

El Engine es el módulo destinado a distribuir las operaciones de cálculo para la factorización de una clave RSA.

En el presente capítulo veremos cómo se encarga de procesar el número a factorizar, dividiendo el problema en sub-problemas con espacio de búsqueda más pequeño, denominados tareas, cómo establece una conexión segura con las máquinas que tenemos a nuestra disposición, cómo distribuye las tareas sobre la red de ordenadores sobre la que trabajaremos (ya sea en el Cloud público de Amazon o nuestra propia red personal) y cómo procesa los resultados generados.

#### 5.3.1.2. Herramientas.

Software utilizado

El Engine es una aplicación que genera y procesa archivos de texto. El lenguaje idóneo para esta labor es **Perl**[27] ("Practical Extraction and Report Languaje"). Perl es un lenguaje de programación de propósito general originariamente desarrollado para la manipulación de texto y usado hoy en día para una amplia gama de tareas, incluyendo administración de sistemas, desarrollo web, programación de redes, desarrollo de interfaces de usuario gráficas (GUI) y mucho más.

Sus características principales son:

- Fácil de usar.
- Soporta tanto programación orientada a objetos como procedimental.
- Potente soporte integrado para procesamiento de texto.

Optamos por implementar el Engine en Perl por su facilidad a la hora de manipular archivos de texto. Otra de las razones de peso es que Perl viene por defecto instalado en las distribuciones más populares de GNU/Linux incluyendo Gentoo, Slackware, Mandriva, Debian, RedHat, SUSE y Ubuntu, entre otros, por lo tanto nos aseguramos que esté instalado en las instancias Amazon que estudiamos.

Software descartado

- **Net::OpenSSH**

Es el paquete de funciones para clientes SSH que proporciona Perl a través del CPAN ("Comprehensive Perl Archive Network"). El CPAN es el repositorio de Perl, en el que se almacenan más de 96.000 módulos escritos en Perl para distintas tareas. Estudiamos su uso en el Engine, pero fue descartado porque no nos permitía liberar al usuario de interactuar con las máquinas remotas en la conexión, es decir, esta biblioteca no cuenta con un método automático para introducir la contraseña de autentificación a la hora de conectar con un servidor.

Librerías adicionales

- **SSHPass**[28]

Utilidad diseñada para ejecutar SSH de forma no interactiva, esto es, sin necesidad de introducir la autentificación cada vez que se realiza una conexión entre dos máquinas remotas. El Engine utiliza la versión 'v1.04'.

Lenguaje para los algoritmos

El algoritmo de la división por tentativa ha sido implementado en C utilizando la biblioteca GMP. El algoritmo de Criba Cuadrática ha sido implementado en C/C++, utilizando la biblioteca NTL integrada con GMP.

[27] http://perldoc.perl.org/index.html
[28] http://sourceforge.net/projects/sshpass/

- **GMP**[29]

Biblioteca para aritmética de precisión arbitraria, operaciones entre números enteros con signo, números racionales y número en coma flotante, necesaria para trabajar con grandes números. El tamaño límite lo impone la cantidad de memoria de la máquina en la que se trabaja. GMP dispone de un conjunto variado y rico de funciones con las que trabajar, las cuales presentan una interfaz regular.

El objetivo principal de GMP es el desarrollo e investigación de aplicaciones criptográficas, aplicaciones de seguridad en Internet, sistemas algebraicos, investigación en álgebra computacional, etc.

GMP ha sido desarrollado para ser lo más rápido posible, tanto para operadores pequeños como muy grandes, siendo la biblioteca más rápida de la actualidad. Su velocidad se alcanza gracias al uso de palabras enteras como tipo aritmético básico, con la implementación de algoritmos asintóticamente rápidos cuyo código ha sido altamente optimizado para la mayoría de CPUs.

GMP es distribuido bajo licencia GNU LGPL[30], por lo que la biblioteca se puede usar, compartir y modificar libremente, permitiendo al usuario hacer públicas sus modificaciones. Esta licencia también impone restricciones a aquellos usuarios que quieran integrar este software con otro software de pago.

El Engine utiliza la versión 'v5.0.1'.

- **NTL**[31]

Biblioteca de alto rendimiento implementada en C++ por Victor Shoup, con la colaboración de otras personas, que provee estructuras de datos y algoritmos para manipular números enteros de longitud arbitraria y con signo, además de matrices, vectores, polinomios sobre enteros y sobre campos finitos. Es perfectamente integrable con GMP, aprovechando así sus características de rendimiento.

En el CD adjunto a esta memoria se encuentran las instrucciones para llevar a cabo la integración de ambas bibliotecas (NTL y GMP) para sistemas tipo Linux.

NTL otorga una interfaz clara y consistente para una gran variedad de objetos matemáticos representados por distintas clases, además de proveer un buen entorno para la implementación fácil y rápida de nuevos algoritmos pertenecientes a la teoría de números, sin sacrificar el buen rendimiento de este.

NTL se distribuye bajo los términos de la GNU General Public License.[32]

El Engine utiliza la versión 'v5.5.2'.

En el Apéndice I se revisa de forma más detallada la implementación de los algoritmos utilizados.

[29] http://gmplib.org/
[30] http://www.gnu.org/copyleft/lesser.html
[31] http://www.shoup.net/ntl/
[32] http://www.gnu.org/licenses/gpl.html

#### 5.3.1.3. SSH: conexión con una máquina con la que no hemos contactado previamente:

Vamos a describir los pasos que realiza el protocolo SSH para conectar con una máquina con la que no hemos tenido una conexión previa. SSH utiliza por defecto para establecer la conexión el puerto 22, por lo que éste deberá de estar abierto en todas las máquinas remotas y en la máquina cliente. Además, el demonio 'sshd' debe estar arrancado en la máquina servidor, esto es, tiene que estar instalado el servidor de SSH en él. Las máquinas de Amazon cuentan con este demonio ejecutándose.

Una posible manera de conectar con una máquina remota mediante SSH es la siguiente:

ssh <usuario>@<IPMáquinaRemota>

'usuario' es el nombre de usuario con el que nos vamos a conectar e 'IPMáquinaRemota' es la dirección IP de la máquina. También se puede utilizar la dirección DNS de la máquina.

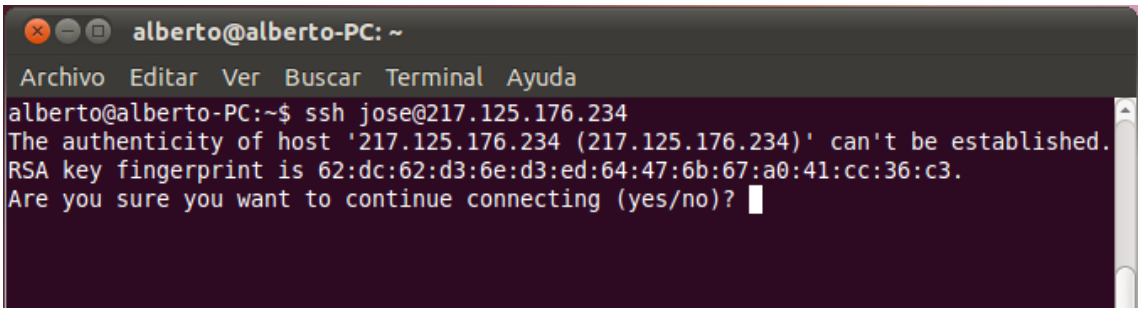


Figura 5.2: Primera conexión (huella digital).

En la imagen 5.2 vemos que SSH nos muestra la "*fingerprint*", o huella digital de la máquina a la que vamos a conectar. Es nuestra responsabilidad contrastarla para verificar que nos estamos conectando al sitio que queremos y no a otro posible sitio malicioso. Nos pregunta que si estamos seguros de que ése es el sitio al que queremos contactar. En nuestro caso lo es, por lo que escribimos "yes".

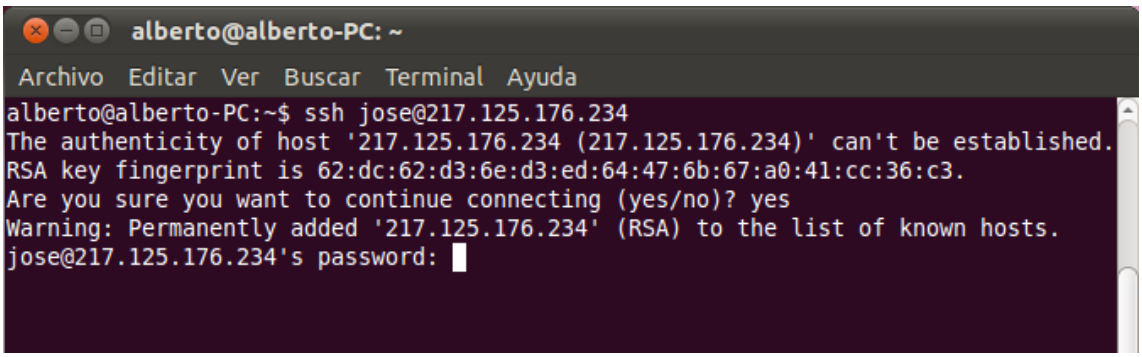


Figura 5.3: Primera conexión (contraseña).

Cuando aceptamos, nuestro cliente SSH guarda la clave del host servidor en el archivo "/$HOME[33]/.ssh/known_hosts". Este archivo es una lista de todas las claves de los hosts que hemos aceptado como válidos. Las entradas de este archivo tienen la siguiente estructura:

1. Uno o más nombres de servidor o direcciones IP, separados por comas.
2. Tipo de la clave: RSA o DSA.
3. Los datos de la clave pública, presentados en formato ASCII.
4. Comentarios (opcionales).

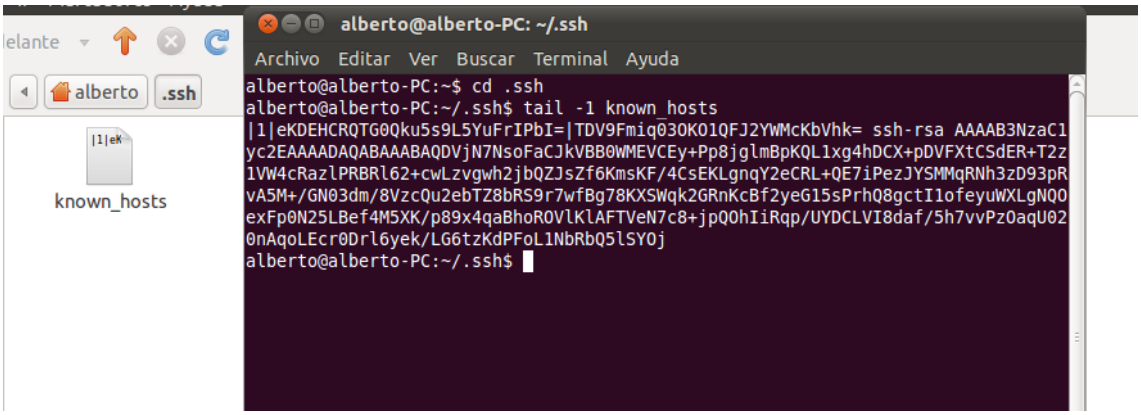


Figura 5.4: Ejemplo de una entrada del archivo "$HOME/.ssh/known_hosts".

Introducimos la contraseña para iniciar la sesión con 'usuario' en la máquina remota y ya estamos conectados con ella, como se puede ver en la Figura 5.5.

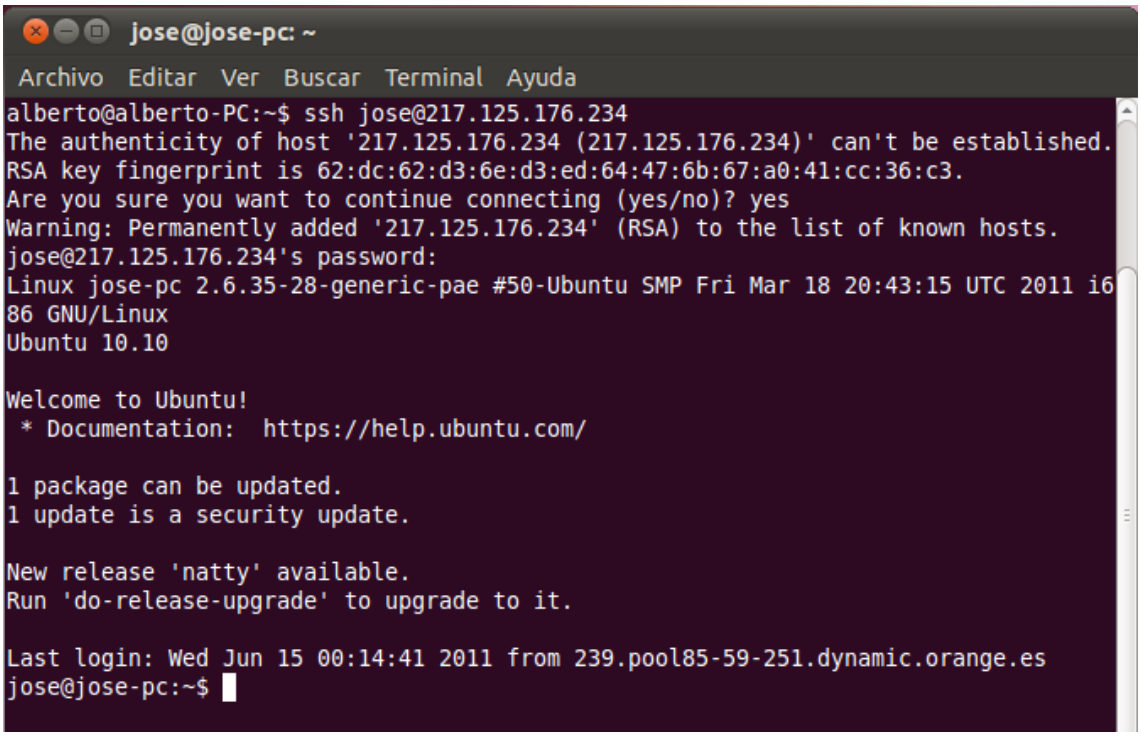


Figura 5.5: Primera conexión (conexión establecida).

[33]$HOME -> /home/usuario, donde 'usuario' representa el nombre de la sesión iniciada en la máquina cliente.

La próxima vez que volvamos a conectarnos al mismo servidor, nuestro cliente SSH nos permitirá saltarnos los pasos de verificación que hemos realizado anteriormente ya que disponemos de la clave del host, alojada en "$HOME/.ssh/known_hosts".

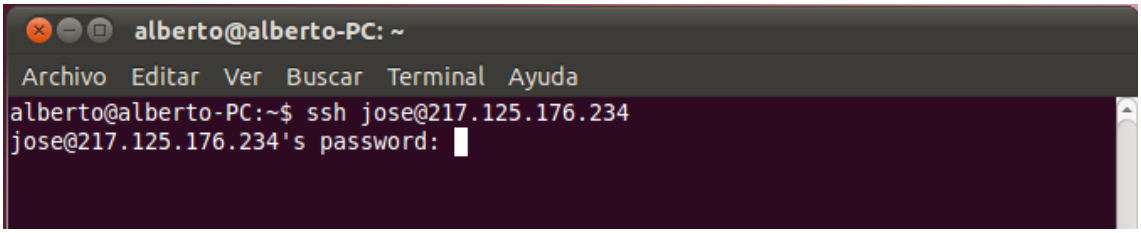


Figura 5.6: Segunda conexión (nueva conexión).

#### 5.3.1.4. Descripción de la arquitectura.

Estructura y funcionalidad.

En cuanto a la estructura, el Engine se divide en dos grandes módulos, atendiendo a la función que se realiza en cada una de ellos:

**Configuración** – Los scripts pertenecientes a este módulo tienen la función de preparar el escenario adecuado para el proceso de factorización. Para ello, se ejecutan los siguientes pasos por orden:

i) Establecer los parámetros – Configurar los tres archivos de la carpeta "$HOME/principal/datos":

- clave.txt – Módulo RSA a factorizar.
- algoritmo.txt – Seleccionar el algoritmo con el que se va a factorizar la clave. Puede seleccionarse 'trialdiv' para la división por tentativa o 'cribacuadratica' para la Criba Cuadrática.
- maquinas.txt – Introducir la información necesaria de las máquinas que van a jugar la función de servidores. El formato del archivo es el siguiente: IP – LIBRE – usuario – contraseña_de_usuario (sin los guiones, separados por espacios). Si una máquina que va a actuar como servidor tiene más de un núcleo y se quiere utilizar, simplemente duplica la línea que contiene la información de dicha máquina.

ii) Crear los rangos de búsqueda – Llamada al script Perl 'creaRangos.pl', encargado de generar los distintos rangos que posteriormente se convertirán en tareas a ejecutar.

iii) Conectar con las máquinas – Llamada al script Perl 'conexión.pl', el cual realiza las operaciones necesarias para establecer la conexión cliente-servidor con todas las máquinas disponibles, cuyos datos se encuentran almacenados en el archivo 'máquinas.txt'. Este script es complejo y fundamental, por lo que pasamos a revisarlo de forma más detallada.

Funcionamiento del script principal del módulo: *'conexión.pl'*

El script 'conexión.pl' es el encargado de establecer una conexión segura y automática, sin interacción por parte del usuario, entre la máquina cliente y las máquinas que actúan con el rol de servidor. Durante este proceso también se copian los archivos necesarios para la ejecución de cada tarea en cada servidor, esto es, el ejecutable del algoritmo y el envoltorio que activa dicho ejecutable. La decisión de qué tipo de ejecutable enviar a cada servidor se toma en función del contenido del archivo 'algoritmo.txt' y del tipo de máquina con la que se vaya a conectar, es decir, de 32 bits o de 64 bits. Esta conexión se consigue con los siguientes pasos:

1º. Generación de la clave pública y eliminación de claves actuales.

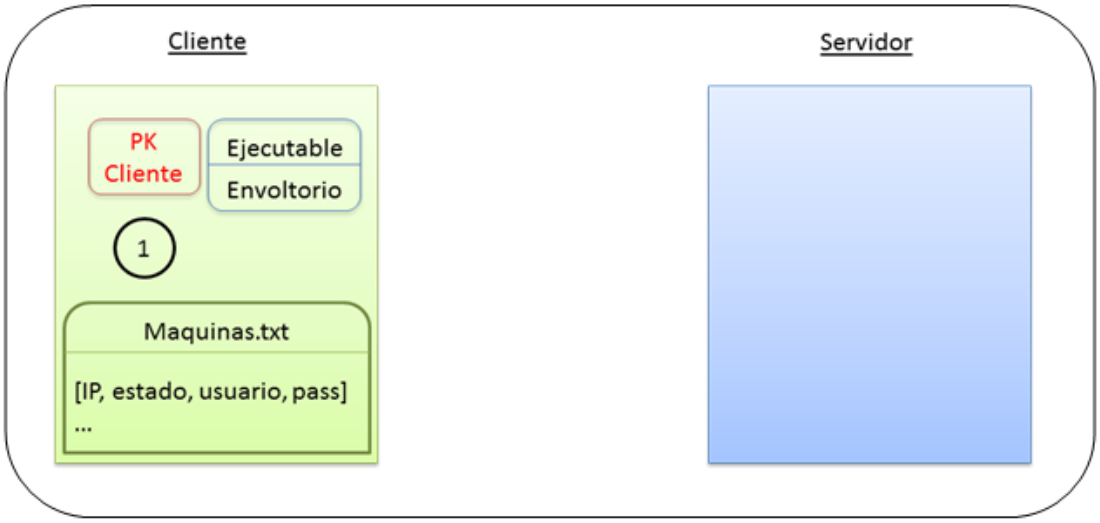


Figura 5.7: Generación de la clave pública y eliminación de claves actuales.

Borramos la carpeta '.ssh', que es la que contiene todas las claves para la sesión actual y los datos de las máquinas a las que el cliente se ha conectado anteriormente. De esta forma evitamos distintas claves para la misma máquina a la que nos vayamos a conectar, solucionando errores del tipo "*too many current keys*" (Demasiadas claves para identificar la misma máquina). Después generamos una clave RSA gracias a la utilidad 'ssh-keygen'[34], con el objetivo de que esta sea copiada en cada uno de los servidores con los que vamos a establecer la conexión segura mediante clave pública. "ssh-keygen" es una utilidad que proveen los sistemas operativos tipo Unix/Linux para generar y administrar claves de autentificación para SSH. Esta herramienta almacena la clave privada generada en el archivo "$HOME/.ssh/id_tipoClave" y la pública en el archivo "$HOME/.ssh/id_tipoClave.pub", donde 'tipoClave' puede ser tanto RSA como DSA. La clave pública será posteriormente copiada a cada máquina remota, de forma que podamos interactuar con dicha máquina sin necesidad de utilizar una contraseña de inicio de sesión SSH.

Los pasos a seguir para generar una clave son los siguientes:

- Elegir el tipo de clave y generarla.

[34]Para consultar más información, acceder al manual tecleando en consola de Linux -> man ssh-keygen.

- Introducir la ruta donde se creará la clave pública (por defecto "$HOME/.ssh/id_tipoClave").
- Introducir una frase que actúe de semilla para la creación de la clave. Esta frase puede dejarse en blanco, pero no se recomienda, debido a que compromete la seguridad de la clave.

La utilidad, para nuestro script, tiene la siguiente sintaxis:

ssh-keygen -t rsa -N '' -f $path/id_rsa

- La opción -t indica el tipo de clave pública que se va a generar (RSA o DSA).
- La opción -N determina la frase que va a ser utilizada para generar la clave. En nuestro caso, para facilitar la no interactividad, la dejamos en blanco y el sistema la genera automáticamente.
- La opción -f indica la ruta del archivo donde se guardará la clave, en nuestro caso "$HOME/.ssh".

2º. Primera conexión, no segura, para comprobar si existe el directorio '.ssh' dentro del directorio de usuario del servidor.

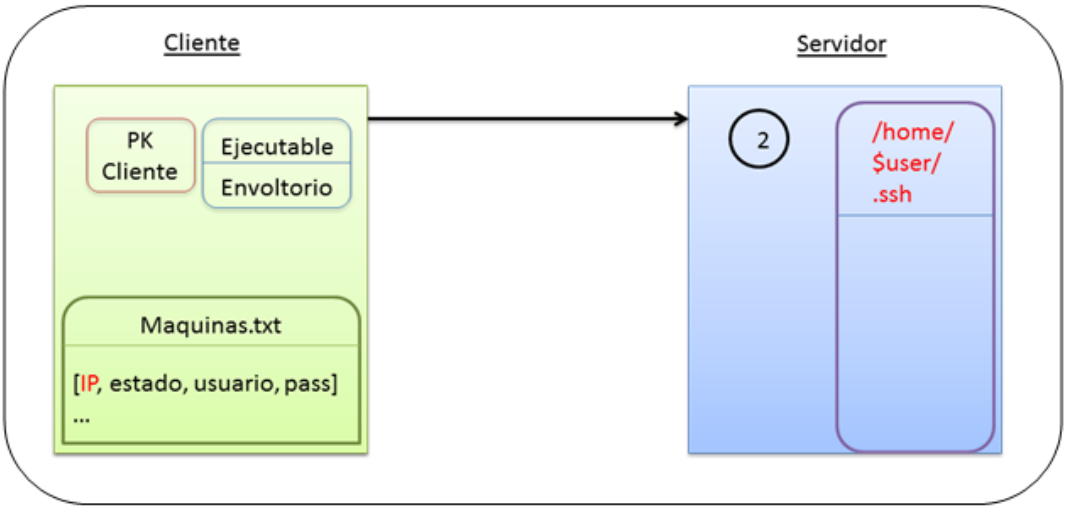


Figura 5.8: Primera conexión.

Leemos la información de cada máquina, alojada en el fichero 'maquinas.txt'. Si hemos introducido más de un núcleo para alguna de ellas, sólo se realizará la conexión una vez para dicho servidor. Comprobamos el directorio de usuario: Si no existe '.ssh', lo crea; si existe, borra su contenido (por la misma razón que explicamos en el punto 1). La importancia de dicho directorio es que SSH trabaja por defecto con las claves contenidas en él. Ésta primera conexión no es segura ya que aún en el servidor no se han generado claves de autentificación SSH, y para poder crearlas, necesitamos que exista el anteriormente citado directorio. Para ello, utilizamos la herramienta 'sshpass', descrita anteriormente, que nos permite acceder a la máquina remota y hacer la comprobación sin que la conexión sea rechazada.

La sintaxis de 'sshpass' es la siguiente:

```
sshpass -p $contraseña ssh -o StrictHostKeyChecking=no
$usuarioRemoto@$IPRemota find /home/$usuarioRemoto -name .ssh
```

- La opción -p permite introducir la contraseña del servidor, de forma que la conexión se realiza de manera automática, no interactiva.
- La opción -o permite modificar las opciones del archivo de configuración de SSH, alojado en /etc/ssh/sshd_config (para la última versión de openssh, para versiones anteriores ver el fichero /etc/ssh/ssh_config). En nuestro caso, establecemos "StrictHostKeyChecking=no" [35] de forma que conectaremos con la máquina 'ciegamente' sin realizar ningún tipo de comprobación de claves y añadiendo la clave del servidor si no está presente a nivel local. Esta opción es necesaria ya que si no deshabilitáramos esta comprobación SSH rechazaría la conexión automática, ya que nunca hemos establecido conexión con ese servidor y queremos prescindir de la interacción de la persona que está ejecutando el script conexión con la máquina remota.
  SSH por defecto tiene asociado el valor de preguntar al usuario a esta variable sobre la veracidad de la conexión. Su objetivo es evitar ataques del tipo 'hombre-en-medio' (*"man-in-the-middle"*), de forma que verificando correctamente el host, no hay forma de que un dispositivo intermedio haya podido estar leyendo o manipulando tus propios paquetes.
- $usuarioRemoto@IPRemota es la dirección del servidor a conectar, junto con el usuario con el que nos identificaremos.
- El resto es el comando que busca dentro de /home/$usuarioRemoto la carpeta llamada .ssh.

3º. Copia clave en servidor. Copia la clave pública del cliente en el servidor, completando la primera parte de la conexión.

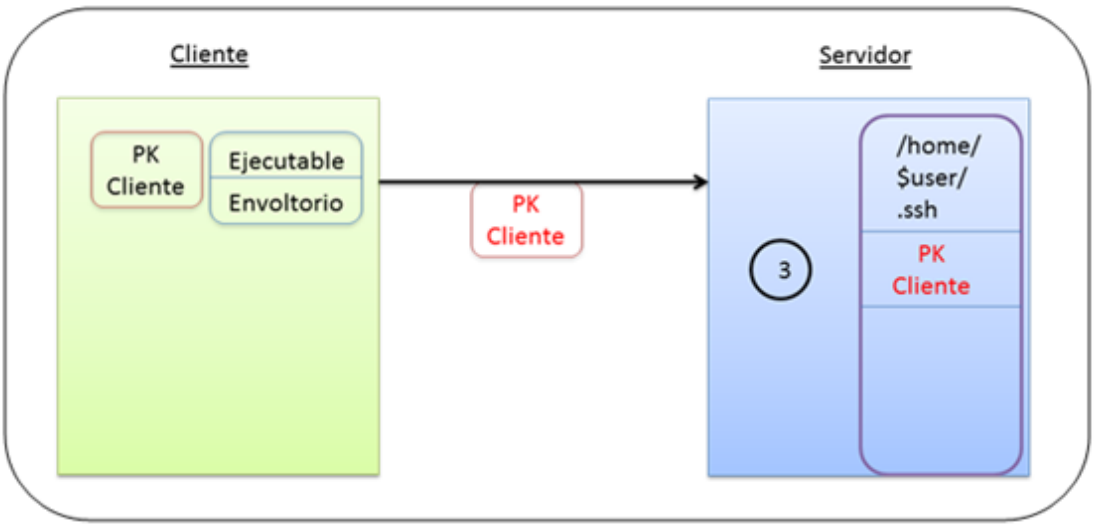


Figura 5.9: Copia la clave pública del cliente en el servidor.

[35] http://www.symantec.com/connect/articles/ssh-host-key-protection

Con esta acción completamos la primera parte del *"handshake"*: el cliente copia su clave pública en cada servidor, de forma que la próxima vez que intente acceder a él su clave será verificada y se asegurará de la validez del host. Para esta acción, utilizamos la herramienta 'ssh-copy-id'[36], que permite instalar nuestra clave pública en el fichero de claves autorizadas del servidor remoto (se copia en el fichero 'authorized_keys' dentro de la carpeta '.ssh').

4º. Lanza agente.

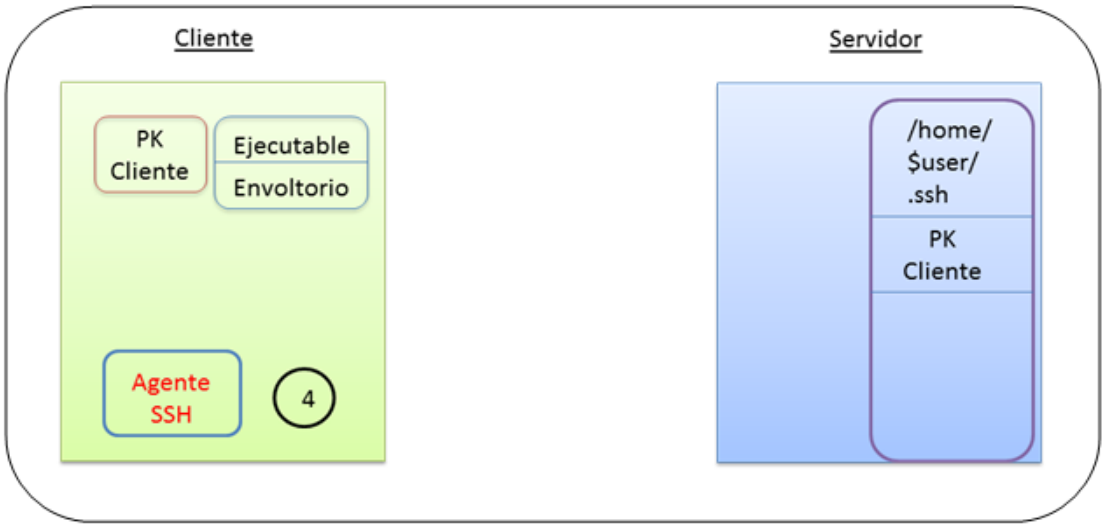


Figura 5.10: Copia la clave pública del cliente en el servidor.

OpenSSH dispone de una herramienta que ayuda al usuario a no tener que recordar la frase con la que se crea la clave pública. Esta herramienta se llama 'ssh-agent'. Para más información sobre cómo actúa el agente, remitimos al lector a [SSHO, capítulo 5]. En este punto activamos el agente SSH para facilitar la no interactividad del script.

5º. Crea la clave pública en el servidor y la copia en el cliente. Creación en cada servidor del archivo 'known_hosts'.

[36]Para consultar más información, acceder al manual tecleando en consola de Linux -> man ssh-copy-id.

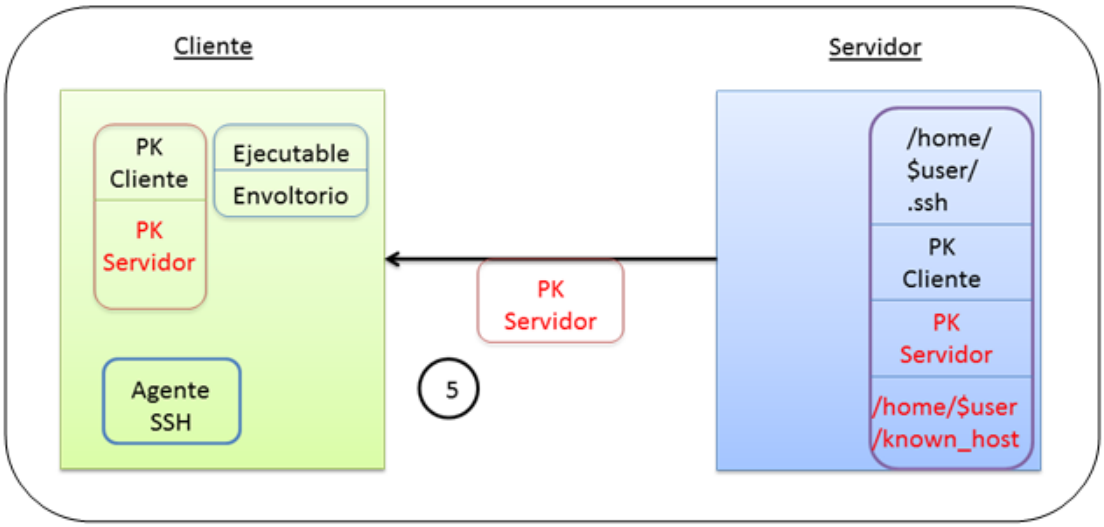


Figura 5.11: Crea la clave pública en el servidor y la copia en el cliente.

Ahora es el turno de que cada servidor remoto haga las mismas acciones que ha llevado a cabo el cliente: cada servidor hace uso de la herramienta 'ssh-keygen' para crear sus propias claves RSA y las copia en el fichero 'authorizated_keys' del cliente, completando así la segunda parte del *"handshake"*. Después se conecta con el cliente para que éste lo verifique como host amigo y lo añada a su fichero 'known_hosts'. Por último ejecuta su agente, completando así el proceso de no interacción.

Cabe resaltar que el cliente obtiene de forma automática su propia IP pública. Esto es necesario para que cada servidor pueda conectarse con el cliente y que se complete el proceso de autentificación. La IP del cliente se obtiene gracias a la herramienta 'wget'[37], que permite la descarga de contenidos desde servidores web de forma sencilla, y que acompañada de la expresión regular Perl adecuada, filtra la información de la página web de la que obtenemos la IP. En nuestro caso obtenemos la IP del servidor 'checkip.dyndns.org'. La sentencia es la siguiente:

wget -q -O - checkip.dyndns.org|grep -o "[0-9]\\{1,3\\}\\.[0-9]\\{1,3\\}\\.[0-9]\\{1,3\\}\\.[0-9]\\{1,3\\}"

[37]http://www.gnu.org/software/wget/

6º. Copia los archivos ejecutables en el servidor: el envoltorio, y el algoritmo de factorización elegido.

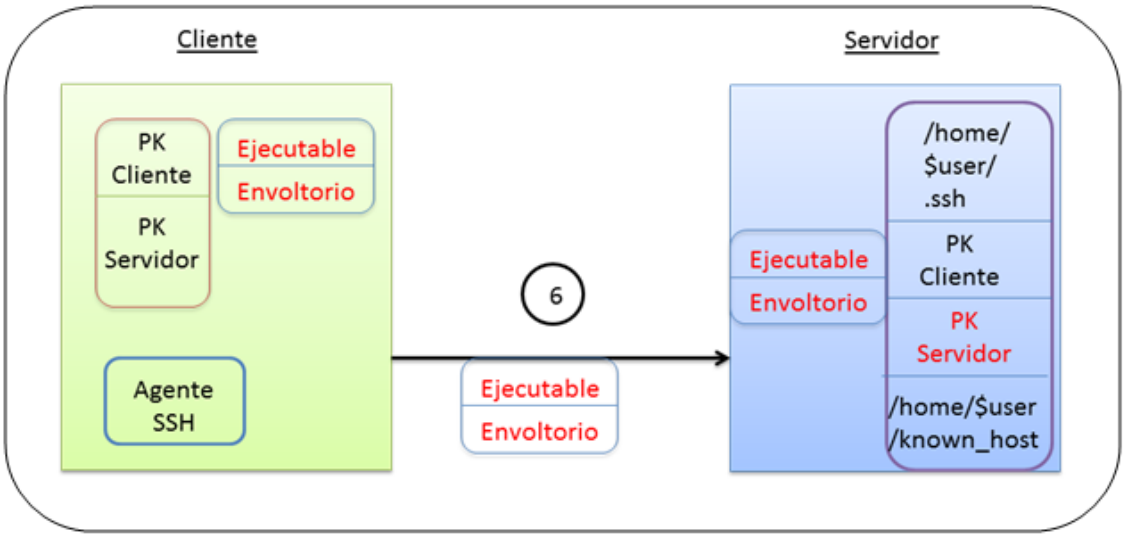


Figura 5.12: Copia los ejecutables en el servidor: envoltorio y algoritmo de factorización.

En la última fase de la conexión, el cliente copia el ejecutable y el envoltorio que activa el ejecutable. Se tiene en cuenta la arquitectura de la máquina remota (32 o 64 bits) y el tipo de factorización escogido, configurado en el archivo 'algoritmo.txt'.

Vamos a revisar ahora brevemente el otro script que actúa en la configuración inicial:

*creaRangos.pl* – Dependiendo del algoritmo elegido para la factorización de la clave, se generan los rangos de dos maneras distintas:

- Creación de rangos para la división por tentativa – Se genera un número de rangos igual al número de máquinas disponible por una constante *'k'*. Esta constante se obtiene después del estudio del comportamiento del algoritmo en la máquina donde se vaya a ejecutar. Para nuestros resultados, $k = 3$. El menor número que puede haber en un rango es dos y el mayor, raíz cuadrada del número a factorizar.
- Creación de rangos para la criba cuadrática – La generación de rangos está centralizada en el cero. La razón es simple, según los principios matemáticos de la criba cuadrática vista en el apartado 4.6, es más probable obtener más elementos válidos cuanto más nos acerquemos a cero, por lo que los rangos tienden a mantenerse lo más cercanos a cero.

Para ambos casos se genera un archivo 'rangos.txt' dentro del directorio "$HOME/principal/datos", posteriormente será utilizado para la creación y distribución de tareas.

**Ejecución** – Los scripts que componen este módulo se encargan de la distribución de las tareas y el procesamiento de los resultados obtenidos por las máquinas remotas. Para ello se ejecuta el script 'tareas.pl', cuya descripción detallamos a continuación:

Primero se inicializa el estado de todas las máquinas a LIBRE, ya que todas se encuentran a nuestra disposición en el momento de la ejecución y no realizan ninguna

otra actividad. Una vez inicializadas las máquinas, ejecuta en segundo plano el monitor de la carpeta de resultados. Éste monitor tiene la labor de revisar periódicamente la carpeta "$HOME/principal/resultados", que es la que contiene todos los archivos de resultados generados por los servidores remotos, y los procesa según el algoritmo de factorización elegido, como veremos más adelante.

Después se sumerge en un bucle en el cual va creando de forma dinámica un archivo con una tarea, de la siguiente forma:

- Crea un archivo llamado 'infoExecContador.txt', donde 'Contador' es el número de la tarea que se va a crear. De esta forma podemos identificar qué tareas está ejecutando cada servidor.
- Extrae un intervalo del fichero 'rangos.txt'.
- Obtiene la clave, la IP del cliente y el usuario del cliente, añadiéndoselo al archivo creado, con el siguiente formato: [clave, intervalo, IP, usuario_Cliente].

Por último, ejecuta el script lanzador, el cual distribuye la tarea generada a la máquina libre correspondiente e inicia el envoltorio que contiene dicha máquina. Este proceso se repite cada vez que haya una máquina libre en el fichero 'maquinas.txt'. El bucle termina al completar todas las tareas generadas para la factorización o cuando se encuentre la factorización del número.

Vamos ahora a describir con más detalle el monitor, y posteriormente revisaremos brevemente el resto de scripts que forman parte de este proceso.

*Monitor.pl*

Como ya hemos comentado antes, la labor del monitor es examinar el directorio donde se almacenan los resultados generados por los servidores y procesar dichos resultados de una forma u otra, dependiendo del algoritmo de factorización elegido previamente en la fase de configuración de parámetros.

La estructura del monitor es sencilla: primero se genera el archivo 'Resultados.txt' dentro de la carpeta de resultados, llamando al archivo ejecutable 'config'. Éste tiene la función de establecer los parámetros de la criba cuadrática necesarios para después llevar a cabo el módulo de álgebra lineal. Si estamos tratando con división por tentativa, no se hace nada con este archivo. Una vez creado este archivo, entra en un bucle que se ejecuta cada cierto periodo de tiempo. Éste tiempo depende del tamaño de la clave: si la clave es pequeña entrará a revisar cada poco tiempo; si por el contrario es grande, se configura para que entre a revisar el directorio en periodos de tiempo más grandes. De esta forma se evita una excesiva carga de trabajo para la máquina cliente. En cada iteración del bucle, obtiene los archivos de la carpeta "$HOME/principal/resultados" y los procesa de la siguiente manera:

- Si se trata de un directorio (directorio actual '.' o directorio padre '..'), se ignora.
- Si se trata del archivo 'activity.xml' (log de resultados del codeswarm, ver sección 5.3.2) o del archivo 'Resultados.txt', se ignora.
- Si se trata de un archivo de resultados recibido de un servidor, se comprueba que no haya sido revisado por el propio monitor previamente. Cada vez que el monitor procesa un archivo, lo renombra con el prefijo "(revisado)",

evitando que sea tratado de nuevo posteriormente. Si no ha sido revisado, dependiendo del tipo de algoritmo de factorización elegido procesamos de distinta forma:

- Si se trata de la división por tentativa, se revisa el archivo para buscar el indicador de si ha sido o no encontrado el factor. Éste indicador lo escribe el propio archivo binario 'trialdiv'. Si la respuesta de esta comprobación es "Sí", extrae el factor encontrado, presenta por pantalla la solución y termina la ejecución. Si no, renombra el archivo y sigue buscando.
- Si se trata de la criba cuadrática, el archivo se procesa contando los elementos que se han encontrado para esa tarea y se almacenan en el archivo 'Resultados.txt'. Cuando se alcanzan los resultados suficientes, se ejecuta el módulo de álgebra lineal. La probabilidad de que se encuentre un factor no trivial para nuestra implementación es de $1023/1024$. Una vez ejecutado el módulo de álgebra lineal, se termina la ejecución.

Vemos ahora el resto de scripts que forman parte de este proceso:

*Lanzador.pl* – Es el encargado de copiar la tarea creada por el script 'tareas.pl' en el equipo remoto y de lanzar el envoltorio en dicha máquina remota.

*Envoltorio.pl* – Su función consiste en ejecutar el algoritmo de factorización adecuado en el servidor donde está alojado, de enviar el archivo resultado generado por el binario al cliente y de comunicar al cliente que la máquina ha terminado su ejecución y por tanto está lista para realizar una nueva tarea.

El nombre del archivo resultado mantiene la siguiente sintaxis:

<Hora>.<minuto>.<segundo>-<día>.<mes>.<año>:<IPMáquinaServidor>

De esta forma se asigna un nombre de archivo unívoco (no pueden haber dos iguales) y nos proporciona los datos necesarios para calcular el tiempo de ejecución de la factorización y saber de qué servidor proviene el archivo.

*ModificaEstado.pl* – Se encarga de modificar el estado de una máquina a OCUPADO, si a la máquina se le ha asignado una tarea y a LIBRE si ha terminado su ejecución.

**Casos de uso: conexión y ejecución del Engine**

- Módulo de configuración

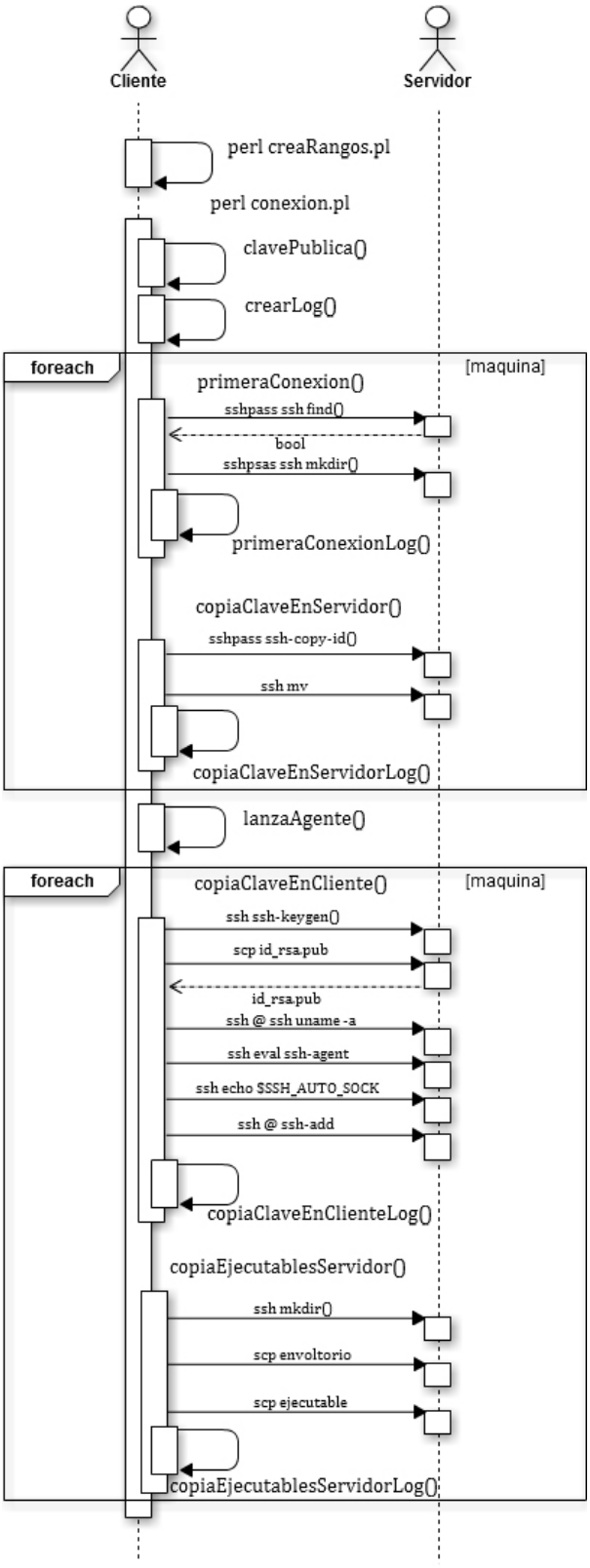


Figura 5.13: Ejecución del módulo de configuración

- Módulo de ejecución

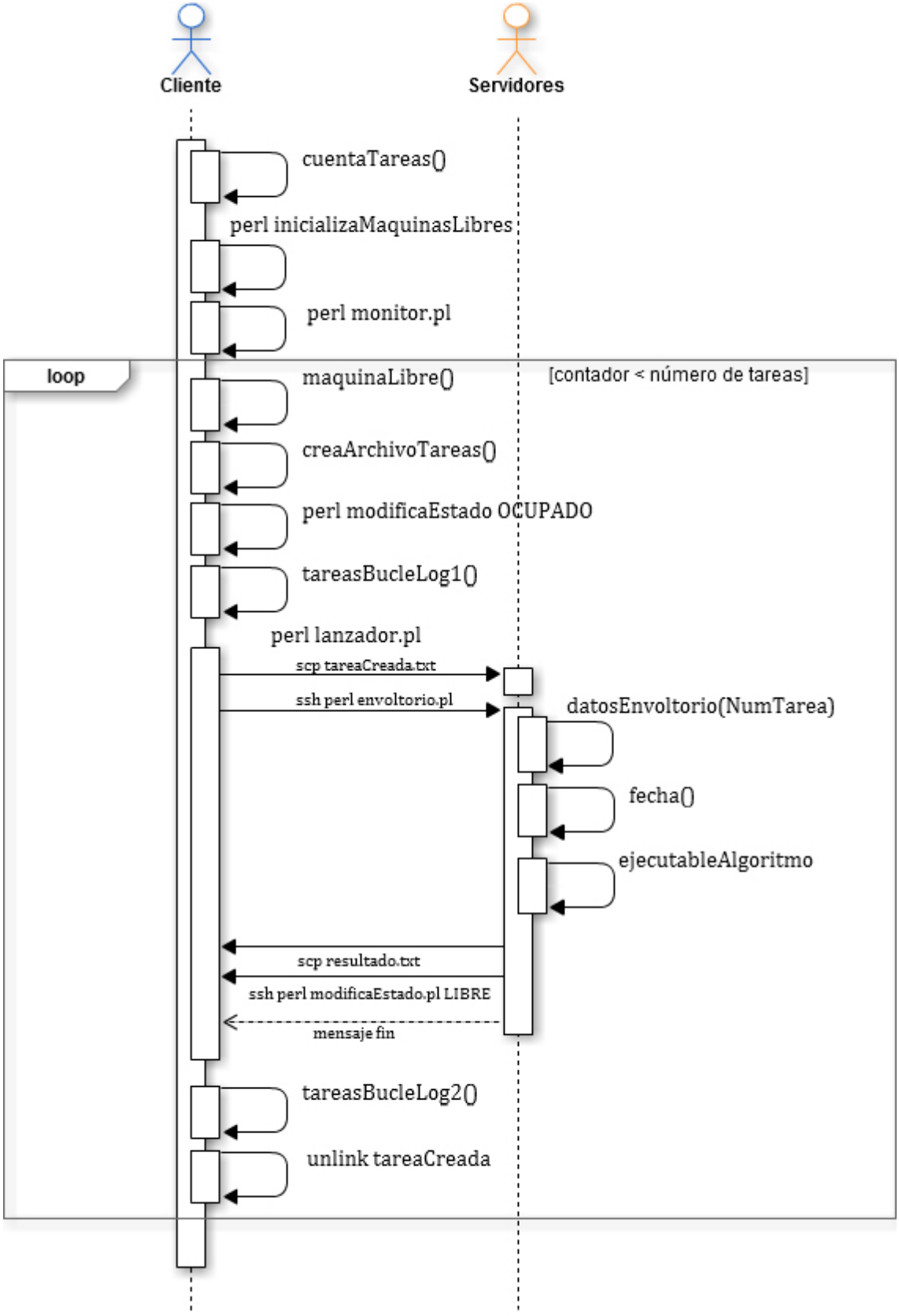


Figura 5.14: Lanzamiento del módulo de ejecución

- Monitor

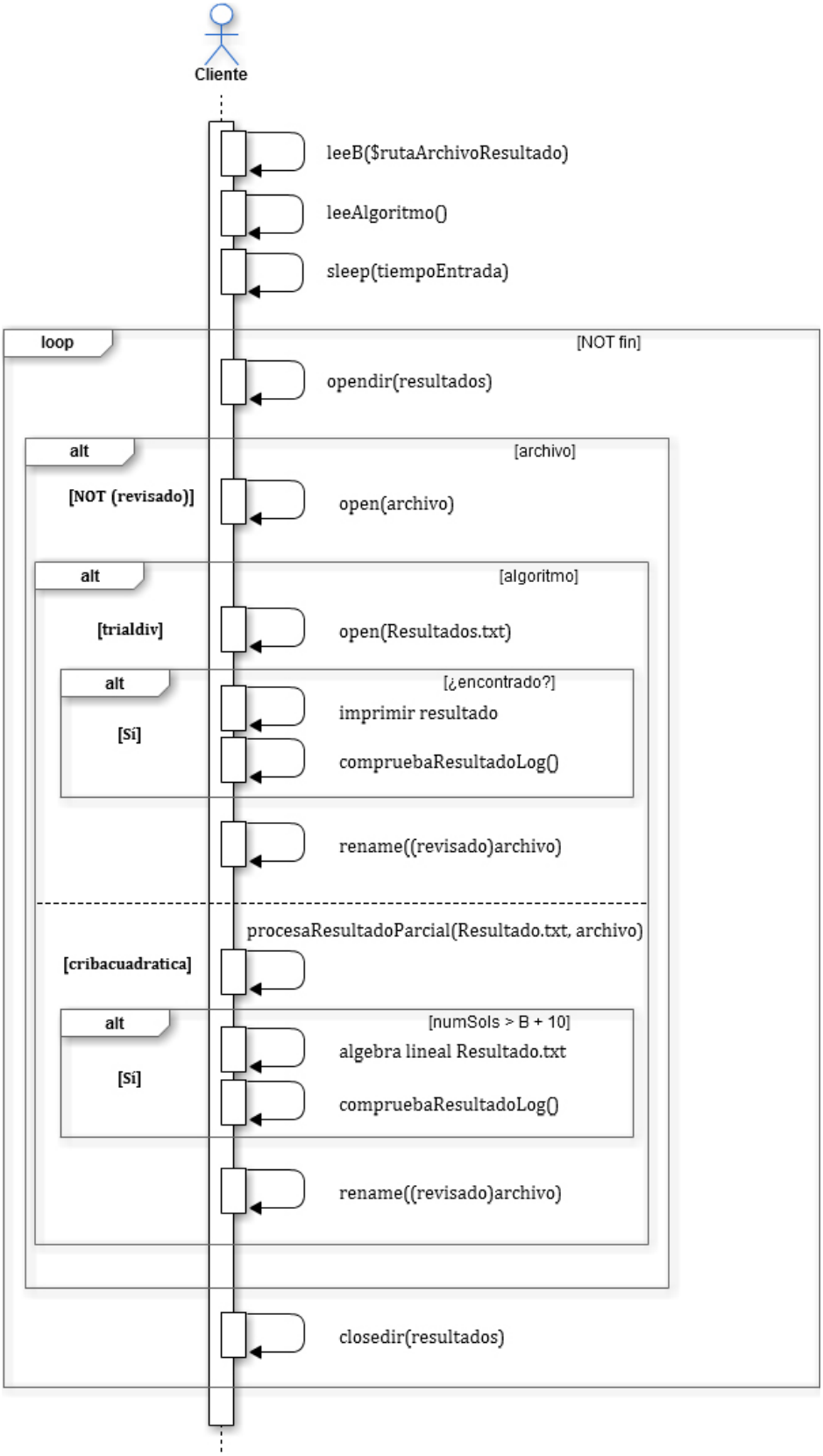


Figura 5.15: Lanzamiento del módulo de ejecución

## 5.3.2 Módulo de representación gráfica de transferencias entre máquinas con Codeswarm

### 5.3.2.1 Introducción

El origen de la inclusión de este módulo en el sistema RSA@Cloud viene dado por la posibilidad de representar visualmente cada fase de ejecución del módulo principal Engine. En las últimas etapas de su desarrollo, el equipo se cuestionaba cómo se podría explicar cómo interaccionan las diferentes máquinas que intervienen en el proceso de cálculo de claves al ejecutarlo.

La intención básica era mostrar como la máquina cliente interaccionaba con el conjunto de servidores asociados a ésta: la primera es básicamente el cerebro o coordinador que se encarga de interpretar un guión en el que es la reparte tareas y recopila resultados, mientras que el resto se encargan de computar el verdadero trabajo de factorización.

En primer lugar, la máquina cliente divide el trabajo total en subtareas, crea archivos con la información necesaria para poder distribuirlas entre los servidores e intercambia claves con los mismos para, en ambas direcciones, poder ejecutar remotamente comandos de forma automática con seguridad mediante protocolo SSH. Acto seguido, transfiere los archivos ejecutables y algoritmos necesarios para factorizar el número. A continuación, los servidores preparan los directorios del sistema. En ese instante, ya están preparados para computar subtareas de trabajo. Durante el proceso distribuido de cálculo de factores, los servidores envían los resultados de las subtareas hasta que uno de ellos genera un archivo de resultado positivo.

La decisión final resultó ser que lo más idóneo sería representar todas y cada una de estas fases del proceso de cálculo donde existen transferencias de archivos, mediante una secuencia visualmente atractiva que describiese de forma gráfica este proceso. Y de aquí la necesidad de encontrar una herramienta o programa que leyese un archivo en el que se guardara un historial de sucesos de este para generar una secuencia de imágenes a partir del mismo que ilustrase los mismos.

Es en este punto donde una aplicación multiplataforma llamada Codeswarm[38] entra a formar parte del sistema. Este software de visualización muestra la historia de los *"commit"* realizados por los diferentes miembros involucrados en un proyecto de software que emplee repositorios como Subversion[39], Mercurial[40] y muchos más. Un repositorio, depósito o archivo es un sitio centralizado, normalmente en Internet, donde se almacena y mantiene información digital, habitualmente bases de datos, archivos informáticos o software en continuo desarrollo, como en el caso de Codeswarm. Se dice que un desarrollador *"hace un commit"* cuando, tras una sesión de programación, realiza cambios en el código o en documentos y los transfiere en el repositorio central del proyecto. En el video que el programa genera tras leer un log, tanto los desarrolladores que han participado en el proyecto como los archivos se representan elementos en continuo movimiento. Cuando un desarrollador *"hace un commit"* de un archivo, éste (representado como un punto luminoso) se enciende y se

[38] http://code.google.com/p/codeswarm/
[39] http://subversion.apache.org/
[40] http://mercurial.selenic.com/wiki/

desplaza hacia ese desarrollador (representado como una cadena de caracteres) y comienza a "orbitar" a su alrededor. Los archivos son de un color u otro de acuerdo a su tipo o extensión, dependiendo de si son código fuente, un archivo de imagen, un documento, etc. Si los archivos o los desarrolladores no han estado activos durante un periodo de tiempo determinado, se desvanecen y desaparecen. Por otra parte, un histograma en la parte inferior de la imagen mantiene una secuencia del volumen de archivos que ha sido transferido anteriormente; cada barra mostrará un tamaño y conjunto diferente de colores dependiendo en función del volumen y tipo de archivos transferido. También, opcionalmente se puede mostrar la fecha y una leyenda que atribuye colores a cada tipo de archivo.

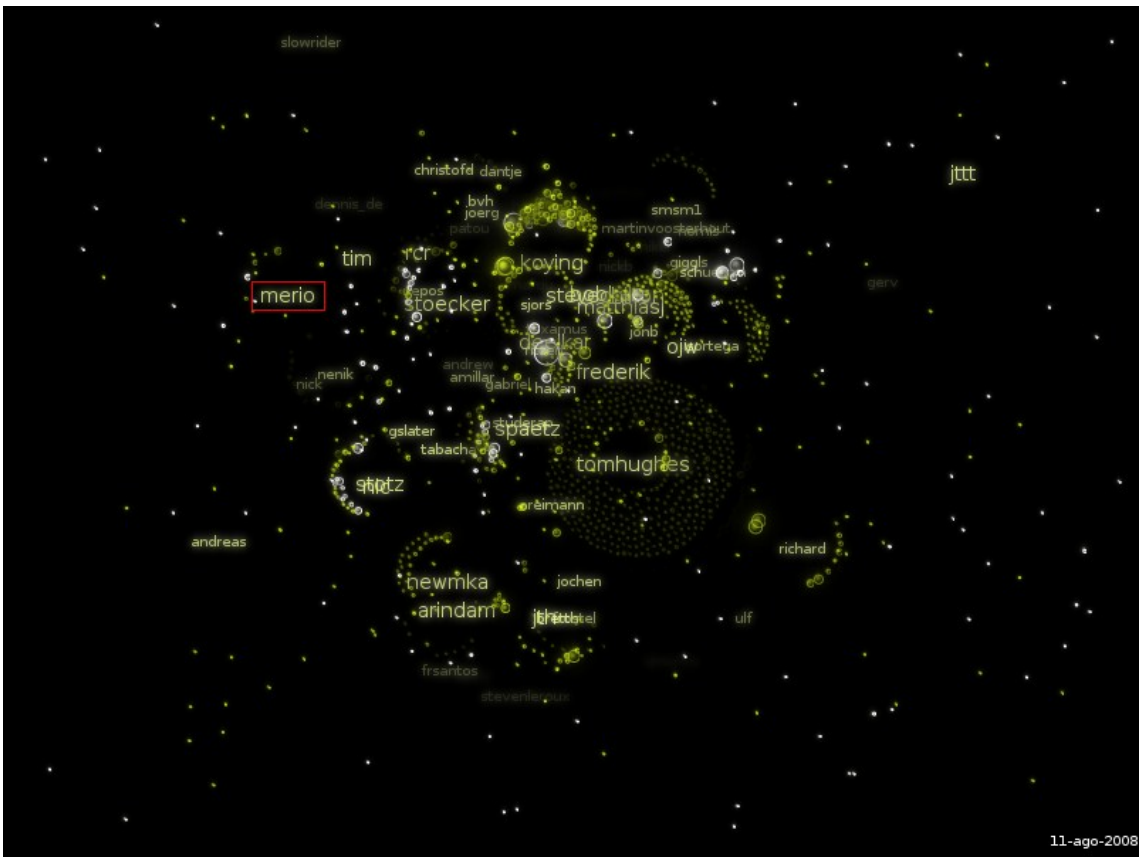


Figura 5.16: Captura de un video generado a partir del fichero log de actividad de un proyecto colectivo coordinado en un repositorio online.[41]

Dado que es una herramienta destinada a generar secuencias de videos de proyectos en repositorios, se ha tenido que adaptar Codeswarm a las necesidades de nuestro sistema. Este apartado escribe el funcionamiento del programa y el proceso llevado a cabo para transformarlo para los propósitos del grupo de proyecto.

**Dependencias**

El único requisito necesario para desarrollar y ejecutar Codeswarm es haber instalado anteriormente una máquina virtual de java JDK y una librería de java

[41] http://www.openstreetmap.org

desarrollada por Apache para compilar y construir programas java mediante un archivo XML como si de un *"Makefile"* se tratara, llamada Ant[42].

Este software es multiplataforma y es posible su ejecución en cualquier sistema UNIX, Windows o Mac OS. De igual modo, las dependencias necesarias descritas en el párrafo anterior también lo son, lo que facilita poder ejecutar Codeswarm en prácticamente casi cualquier máquina.

Una vez conocida la estructura y funcionamiento de Codeswarm, fue necesario adaptar este software a nuestras necesidades de tal modo que representase gráficamente transferencias de archivos entre diferentes máquinas en lugar de actualizaciones de repositorios. Para ello hubo que establecer analogías entre ambos tipos de eventos.

#### 5.3.2.2 Logs de transferencia: Proceso de generación y estructura

El primer punto al que se concluye tras el estudio de este programa es que el módulo principal Engine debería de ser ampliado de modo que la máquina cliente fuese capaz de generar en tiempo real de ejecución un archivo que funcionase como una traza o archivo log que almacenase toda la información relativa a la transferencia de archivos. A partir de este punto de partida, fue necesario estudiar el formato y estructura que los archivos log de Codeswarm son capaces de generar a partir de cierta información contenida en los repositorios.

En efecto, de los repositorios se puede extraer el historial contenido en archivos con extensión '.log'. Codeswarm incluye un conjunto de scripts en Python que permiten transformarlo a XML, los archivos con esta extensión son leídos por el programa para generar una secuencia de video. Por tanto, el módulo Engine debe generar un archivo de este formato y estructura:

```
<?xml version="1.0" ?>
<file_events>
        <!—CUERPO del archivo.Esta línea es un comentario. -->
</file_events>
```

El cuerpo del archivo generado está formado por sentencia que representan eventos o modificaciones en el repositorio. Todas son de la siguiente forma:

```
<event
        date="0"
        filename="/src/helloworld.c"
        author="programmer"
/>
```

[42]http://ant.apache.org/

La etiqueta 'date' informa de la fecha en el formato de UNIX expresado en milisegundos, la etiqueta 'filename' indica la ruta completa del archivo perteneciente al repositorio que ha sido modificado y la etiqueta 'author' corresponde al nombre del miembro del repositorio que ha realizado tales modificaciones. Opcionalmente puede incluirse otra etiqueta en la que se indica el peso o relevancia del archivo modificado en el conjunto del proyecto, que Codeswarm interpreta de tal modo que atribuye al punto luminoso correspondiente al archivo designado un determinado tamaño. Esta relevancia se indica con números enteros y es totalmente relativa, es decir, queda a la elección del usuario determinar el tamaño de los puntos luminosos. Por ejemplo, para asignar a un archivo un peso con valor 1000, añadimos a una línea como la anterior:

```
weight="1000"
```

Otra característica importante de esta etiqueta es que, a medida que se atribuye constantemente un valor para este parámetro cada vez que un archivo es referenciado en el archivo log, el punto luminoso correspondiente con dicho archivo aumentará de tamaño progresivamente. En resumen, cuanta más relevancia tenga en la etiqueta 'weight' y más se referencie un archivo, mayor será su punto luminoso.

**Analogías entre los eventos de un repositorio online y el módulo Engine**

Una vez sabemos qué representan las sentencias incluidas en los archivos log XML generados por Codeswarm para representar la actividad de un repositorio, debemos establecer una analogía de la misma con la ejecución del módulo Engine y las transferencias de archivos entre máquinas que ello conlleva. Los dos eventos básicos que queremos representar visualmente con Codeswarm son básicamente la creación o modificación de ficheros y la transferencia entre máquinas de los mismos, y pueden recrearse de la siguiente manera:

- **Creación/modificación de archivo**: Cuando un miembro de un repositorio crea o modifica un archivo, se genera una línea como la mostrada de ejemplo anteriormente. Codeswarm interpreta esta sentencia de tal forma que, durante la secuencia de video generada, aparecerá un punto luminoso que se dirigirá hacia la cadena de caracteres que representa el nombre del programador involucrado en la creación de ese archivo. Si el archivo fue creado o modificado recientemente, este punto continuará girando en la "órbita" del desarrollador correspondiente hasta cierto tiempo definido por el usuario. Este suceso será equivalente a la creación o modificación de un archivo por parte de una máquina durante la ejecución del módulo principal Engine. Mostremos un ejemplo:

```
<event
        date="1297609933000"
        filename="/resultados/activity.xml"
        author="Cliente"
        weight="10"
/>
```

En este caso, la creación o modificación del archivo aconteció el 15 de febrero de 2011 a las 15 horas, 12 minutos y 13 segundos. Su ruta relativa al directorio de trabajo era "$home/resultados/activity.xml", la máquina en la que se creó o modificó fue 'Cliente', es decir, la máquina encargada de gestionar la ejecución del módulo Engine y a este archivo se le atribuye un peso o relevancia de 10.

**Transferencia de archivo**: Para los casos en los que, durante la ejecución paralela del algoritmo de factorización, una máquina (ya sea el cliente o cualquier máquina remota) transfiere un archivo a otra, la forma más adecuada para representar estos eventos sería que un punto luminoso se desplazase de una "órbita" correspondiente a una máquina a otra. Éste suceso es completamente análogo a la siguiente sucesión de eventos que acontecen durante el desarrollo de un proyecto a un repositorio en el que se efectúa ese desplazamiento: en primer lugar, un archivo es creado o modificado por un miembro del grupo y, al cabo de un tiempo, otro programador modifica, amplia o corrige ese archivo. Dicho de otro modo, si encontramos el siguiente código en archivo log, con Codeswarm se interpretará como una transferencia de archivos:

```
<event
        date="1294752926000"
        filename="192.168.180.129:/home/jose/principal/datos/infoExec1.txt"
        author="Cliente"
        weight="20"
/>
<!— (...) Un número indeterminado de sentencias. -->
<event
        date="1294752927000"
        filename="192.168.180.129:/home/jose/principal/datos/infoExec1.txt"
        author="192.168.180.129"
        weight="20"
/>
```

La primera sentencia se refiere a la creación en la máquina 'Cliente' de un archivo de texto con nombre 'infoExec1.txt', archivo destinado a ser transferido al directorio "/home/jose/principal/" de la máquina con dirección IP 192.168.180.129. La creación

data del día 11 de enero de 2011 a las 13 horas, 35 minutos y 26 segundos. Este archivo es de gran relevancia porque será uno de los archivos utilizados por la máquina con dicha IP para leer la información requerida para ejecutar una subtarea o intervalo parcial del algoritmo de factorización de la clave a descifrar, y por ello tiene un peso o relevancia 20. Un segundo más tarde, según se puede observar en la etiqueta 'date' de la segunda sentencia, este archivo es transferido a la máquina 192.168.180.129 tal y como indica la etiqueta 'author' en el directorio indicado en 'filename'.

Como puede observarse, la interpretación de estos dos tipos de eventos con estas sentencias basta para representar todas las interacciones posibles de archivos por parte de las máquinas.

Ahora bien, antes de comenzar a estudiar la implementación de las directrices necesarias para crear un archivo log en tiempo real mediante la creación de funciones que lo generen dentro de los scripts Perl que conforma el módulo Engine, es conveniente estudiar el funcionamiento interno de Codeswarm para conocer qué opciones y posibilidades nos ofrece este programa para establecer una configuración que se ajuste a nuestras preferencias.

#### 5.3.2.3 Configuración y funcionamiento de Codeswarm

El objetivo de emplear esta herramienta era el de conseguir representar la interacción entre máquinas durante la ejecución del sistema de la mejor manera posible, esto es, generando un video visualmente atractivo y que fuese ilustrativo al mismo tiempo. Por estas razones, es importante estudiar las posibilidades que nos ofrece Codeswarm.

Dentro del directorio donde se encuentra el programa en nuestro sistema de archivos, se observa que el programa está dividido en diferentes módulos: la carpeta 'src' contiene el código en Java de la aplicación, 'lib' contiene las librerías que emplea dicho código, en 'build' se guardan los objetos generados al compilar el código mediante Ant, 'dist' aloja un archivo '.jar' de la aplicación, 'convert_logs' es el módulo de scripts en Python que transforma archivos .log en formato XML, 'physics_engine' almacena las diferentes configuraciones de los motores de "física" aplicada a los elementos representados durante la ejecución para determinar su comportamiento, de los cuales se elige uno en un fichero de configuración de la carpeta 'data', en la cual se configuran multitud de parámetros que veremos a continuación. En esta ubicación también debemos pegar el archivo de trazas XML con el historial de eventos generado anteriormente que queremos representar gráficamente. En el directorio raíz de Codeswarm encontramos archivos como el 'buid.xml' que permite compilar el programa, un LÉAME, los ejecutables para todas las plataformas, un archivo de proyecto para ser abierto y editado con Eclipse y otro archivo referente a los derechos de copia y distribución (éste programa es software libre).

En el directorio 'data', donde debe localizarse el archivo de configuración con extensión '.CONFIG', es donde debemos enfocar toda nuestra atención para configurar multitud de parámetros que se ajusten a nuestras preferencias a la hora de determinar cómo debe ser la representación gráfica de lo acontecido al ejecutar el Engine. Para describir todas y cada una de la opciones posibles, se mostrará por partes el contenido del archivo de configuración que Codeswarm trae por defecto para ir analizando los parámetros y los valores que pueden tomar. Los comentarios en este tipo se archivos van precedidos del carácter almohadilla "#". Usaremos esta circunstancia para las

explicaciones pertinentes con un formato distinto (en azul oscuro y en negrita). Este archivo está nombrado como 'sample.config' (ver Apéndice II).

Por último, es importante señalar que se puede modificar el código fuente localizado en la carpeta 'src'. Se pueden implementar las modificaciones directamente sobrescribiendo los archivos necesarios y compilando el programa mediante el comando "ant" desde el directorio donde esté ubicado Codeswarm o a través de una herramienta de programación como Eclipse o NetBeans. En nuestro caso, únicamente se modificó un valor dentro del método llamado "drawHistogram()" en el archivo 'code_swarm.java' para aumentar el grosor de la línea del histograma.

**Integración con el Engine: generación en tiempo real de logs**

Una vez estudiada la estructura interna y el funcionamiento de Codeswarm, es posible implementar las ampliaciones necesarias para que el módulo principal Engine genere un archivo de trazas o log en tiempo de ejecución.

La máquina cliente es la única que interviene en todas las comunicaciones existentes, ya que el sistema funciona sobre una red radial o en estrella de computadores donde, al contrario de lo habitual, el cliente es el nodo central que envía información al resto de máquinas, que actúan como servidores, es decir, reciben datos, los procesan y envían una respuesta. No es necesario que los servidores puedan comunicarse entre ellos: los archivos que recibe el nodo central son los resultados parciales de cada subtarea de la factorización; cuando se ha recopilado la información necesaria la máquina cliente decide suspender toda comunicación y da por finalizada la ejecución. Concretando, podemos decir que esta máquina actúa como monitor de la ejecución, como gestor del envío de instrucciones y subtareas a los servidores conectados a ella así como de receptor y analizador de los resultados finales.

La máquina cliente siempre es un extremo de toda comunicación y posee toda la información existente relativa a la tarea principal del Engine y al resto de las máquinas. Estos hechos justifican que el archivo log se genere y actualice en la máquina cliente. Los procesos en los que existe creación, modificación o envío de datos son:

- 1: Cuando se realiza conexión con los servidores.
- 2: En la posterior copia de los archivos necesarios para ejecutar subtareas en los servidores.
- 3: Durante la creación de la información necesaria para la ejecución de cada subtarea que les asigna.
- 4: Al enviar estos datos a dichos servidores.
- 5: En el momento en el que un servidor crea un archivo resultado al finalizar el cómputo de una subtarea ...
- 6: ... y acto seguido la máquina cliente los recibe.
- 7: Además esta máquina está constantemente comprobando si la información obtenida es necesaria para dar por finalizada la ejecución y suspender toda comunicación.

Figura 5.17: Topología de red: el sistema crea una red en estrella atípica, donde la máquina donde se ejecuta (nodo central) no es un servidor, sino que es un cliente que establecerá una comunicación con un número determinado de servidores con el fin de enviarles peticiones, que serán respondidas por los mismos tras ser procesadas. Así mismo, no es necesaria la comunicación entre servidores.

Todos estos eventos son seguidos por la máquina cliente en los siguientes scripts del módulo Engine:

- *conexión.pl* (eventos 1 y 2): Este script es el primero en ser ejecutado para establecer una comunicación bidireccional y automática entre cliente y servidores. También copia los scripts y ejecutables necesarios para computar subtareas en los servidores.
- *tareas.pl* (eventos 3, 4,5 y 6): Arranca el componente principal del Engine, encargado de gestionar las comunicaciones, y chequea si el estado de los servidores (libre u ocupado).
- *monitor.pl* (evento 7): El monitor del Engine es invocado en segundo plano por el script tareas, y comprueba permanentemente si se han obtenido datos suficientes para generar un resultado final y finalizar la ejecución del sistema.

A continuación se mostrarán qué funciones son llamadas por el sistema en cada uno de éstos scripts con el fin de completar el archivo log de la manera más fiel posible a los hechos acontecidos en la realidad durante toda la ejecución del módulo. El orden en el que son presentados es el seguido por la enumeración de eventos anterior.

La información proporcionada para cada función será:

- Nombre de la función y script en la que se implementa.
- Evento al que pertenece.
- Parámetros de entrada.
- Breve descripción de su funcionamiento.

## conexion.pl

**A)**

**Nombre de la función**: crearLog

**Evento al que pertenece**: Evento 1

**Parámetros de entrada**:

- $path: Ruta de la forma "/home/[nombre_de_usuario_del_sistema]/.ssh" en la máquina cliente. Este directorio es el que emplea el protocolo SSH por defecto la clave RSA que se emplea para cifrar comandos SSH salientes, las claves públicas externas autorizadas y la lista de *"hosts"* conocidos.
- $subPath: Directorio de usuario de sistema "/home/[nombre_de_usuario_de_sistema]" en la máquina cliente.

**Descripción**:

Invocada al conmienzo del script 'conexión.pl'. Se crea el archivo log XML 'activity.xml' ubicado en "$subPath/principal/resultados", el directorio donde se guardan los archivos que el módulo Engine produce como *"output"*. Se escribe la cabecera XML propia de este tipo de archivos, para más registrar los primeros eventos concernientes a la creación de las claves RSA que empleará la máquina cliente.

**B)**

**Nombre de la función**: primeraConexionLog

**Evento al que pertenece**: Evento 1

**Parámetros de entrada**:

- $subPath: Directorio de usuario de sistema "/home/[nombre_de_usuario_de_sistema]" en la máquina cliente.
- $datos0: IP de la máquina remota con la que se conecta la máquina cliente.
- $datos2: Nombre de usuario del sistema de la máquina remota.

**Descripción**:

Invocada en la función "primeraConexión", se recrea en archivo log el hecho de que la máquina cliente haya ordenado a la máquina remota con IP $datos0 crear sus propias claves RSA en la carpeta "/home/$datos2/.ssh", que usará para conectarse a la propia máquina cliente.

**C)**

**Nombre de la función**: copiaClaveEnServidorLog1

**Evento al que pertenece**: Evento 1

**Parámetros de entrada**:

- $subPath: Directorio de usuario de sistema "/home/[nombre_de_usuario_de_sistema]" en la máquina cliente.
- $claveRSA: Ruta de la forma "/home/[nombre_de_usuario_del_sistema]/.ssh" en la máquina cliente. Este directorio es el que emplea el protocolo SSH por defecto la clave RSA que se emplea para cifrar comandos SSH salientes, las claves públicas externas autorizadas y la lista de *"hosts"* conocidos.
- $datos0: IP de la máquina remota con la que se conecta la máquina cliente.
- $destino: Ruta "/home/[nombre_de_usuario_del_sistema]/.ssh" en la máquina remota.

**Descripción**:

Invocada en la función "copiaClaveEnServidor", queda registrado en el archivo log el evento referente a la copia de la clave pública de la máquina cliente en $claveRSA a la lista de claves autorizadas de la máquina remota ubicada en $destino.

**D)**

**Nombre de la función**: copiaClaveEnServidorLog2

**Evento al que pertenece**: Evento 1

**Parámetros de entrada**:

- $subPath: Directorio de usuario de sistema "/home/[nombre_de_usuario_de_sistema]" en la máquina cliente.
- $datos0: IP de la máquina remota con la que se conecta la máquina cliente.
- $destino: Ruta "/home/[nombre_de_usuario_del_sistema]/.ssh" en la máquina remota.

**Descripción**:

Invocada en la función "copiaClaveEnServidor", se toma nota en el archivo log que se renombrado el nombre del archivo del servidor remoto con la lista de las claves públicas autorizadas en la ruta $destino para hacerlo compatible con la versión 2.0 del protocolo SSH.

**E)**

**Nombre de la función**: copiaClaveEnClienteLog1

**Evento al que pertenece**: Evento 1

**Parámetros de entrada**:

- $subPath: Directorio de usuario de sistema "/home/[nombre_de_usuario_de_sistema]" en la máquina cliente.
- $claveRSA: Ruta de la forma "/home/[nombre_de_usuario_del_sistema]/.ssh"

en la máquina cliente. Este directorio es el que emplea el protocolo SSH por defecto la clave RSA que se emplea para cifrar comandos SSH salientes, las claves públicas externas autorizadas y la lista de "hosts" conocidos.
- $datos0: IP de la máquina remota con la que se conecta la máquina cliente.
- $destino: Ruta "/home/[nombre_de_usuario_del_sistema]/.ssh" en la máquina remota.

**Descripción**:

Invocada en la función "copiaClaveEnCliente", queda registrado en el archivo log el evento referente a la copia de la clave pública de la máquina remota en $datos0 a la lista de claves autorizadas de la máquina cliente ubicada en $claveRSA.

**F)**

**Nombre de la función**: copiaClaveEnClienteLog2

**Evento al que pertenece**: Evento 1

**Parámetros de entrada**:

- $subPath: Directorio de usuario de sistema "/home/[nombre_de_usuario_de_sistema]" en la máquina cliente.
- $claveRSA: Ruta de la forma "/home/[nombre_de_usuario_del_sistema]/.ssh" en la máquina cliente. Este directorio es el que emplea el protocolo SSH por defecto la clave RSA que se emplea para cifrar comandos SSH salientes, las claves públicas externas autorizadas y la lista de *"hosts"* conocidos.
- $datos0: IP de la máquina remota con la que se conecta la máquina cliente.
- $destino: Ruta "/home/[nombre_de_usuario_del_sistema]/.ssh" en la máquina remota.

**Descripción**:

Invocada en la función "copiaClaveEnCliente", actualiza el archivo log al incluir las líneas de evento referentes a la creación en la máquina remota del archivo 'known_hosts' en $destino, al que se añade la información de la máquina cliente en $claveRSA para permitir futuras conexiones. Se completa la conexión automática bidirección entre cliente y servidor.

**G)**

**Nombre de la función**: copiaEjecutablesServidorLog

**Evento al que pertenece**: Evento 2

**Parámetros de entrada**:

- $subPath: Directorio de usuario de sistema "/home/[nombre_de_usuario_de_sistema]" en la máquina cliente.
- $datos0: IP de la máquina remota con la que se conecta la máquina cliente.

- $destinoEjec: Ruta "/home/[nombre_de_usuario_del_sistema]/principal" en la máquina remota. Tanto en la máquina cliente como en los servidores, este será el directorio raíz de trabajo.

**Descripción**:

Invocada en la función "copiaEjecutablesEnServidor", se registra en el archivo log la copia del envoltorio y el ejecutable del algoritmo empleado para la tarea de factorización en $destinoEjec, dentro de la máquina remota con IP $datos0.

## tareas.pl

**A)**

**Nombre de la función**: tareasInicioLog

**Evento al que pertenece**: Evento 3

**Parámetros de entrada**:

- $subruta: Directorio de usuario de sistema "/home/[nombre_de_usuario_de_sistema] /principal" en la máquina cliente, es decir, el directorio raíz de trabajo.

**Descripción**:

Invocada al comienzo del script 'tareas.pl'. Se actualiza el archivo log con los siguientes sucesos: asignación del estado LIBRE a todas las máquinas remotas listadas en el archivo de texto 'máquinas.txt' en la ruta "$subruta/resultados" de la máquina cliente mediante el script 'modificaestado.pl' e inicio de 'monitor.pl' (ver log de este script más adelante).

**B)**

**Nombre de la función**: tareasBucleLog1

**Eventos a los que pertenece**: Eventos 3 y 4

**Parámetros de entrada**:

- $subruta: Directorio de usuario de sistema "/home/[nombre_de_usuario_de_sistema] /principal" en la máquina cliente, es decir, el directorio raíz de trabajo.
- $rutalanzador: Ruta del script 'lanzador.pl' en la máquina cliente, que manda arrancar el ejecutable del algoritmo de factorización en el servidor.
- $maquina: dirección IP de la máquina remota.
- $contador: número de archivo de información de tarea creado por la máquina cliente para ser enviado al servidor.

**Descripción**:

Invocada en el bucle principal del script 'tareas.pl'. El archivo log registra la creación en "$subruta/datos/" del archivo de datos con la información necesaria para que la máquina remota ejecute la siguiente subtarea. A continuación el cliente ejecuta el lanzador en $rutalanzador para que dicho servidor con IP $maquina reciba ese archivo y compute la subtarea con el ejecutable correspondiente. Se asigna el estado "OCUPADO" a la máquina remota en el archivo de texto de la máquina cliente llamado 'máquinas.txt' en la ruta "$subruta/resultados" mediante el script 'modificaestado.pl'.

**C)**

**Nombre de la función**: tareasBucleLog2

**Evento al que pertenece**: Eventos 5 y 6

**Parámetros de entrada**:

- $subruta: Directorio de usuario de sistema "/home/[nombre_de_usuario_de_sistema] /principal" en la máquina cliente, es decir, el directorio raíz de trabajo.
- $maquina: dirección IP de la máquina remota.
- $contador: número de archivo de información de tarea creado por la máquina cliente para ser enviado al servidor.

**Descripción**:

Invocada en el bucle principal del script'tareas.pl'. Queda constancia en el archivo log de los eventos concernientes al creación y envío de los resultados de la ejecución de la subtarea por parte del servidor con IP $maquina. Tras ejecutarse el algoritmo, la máquina remota crea un archivo resultado en $subruta que será enviado al directorio "$subruta/resultados" de la máquina cliente. Por último, se asigna el estado LIBRE a la máquina remota en el archivo de texto 'máquinas.txt' en la ruta "$subruta/resultados" mediante el script 'modificaestado.pl', ambos ubicados en la máquina cliente.

## monitor.pl

**A)**

**Nombre de la función**: compruebaResultadoLog

**Evento al que pertenece**: Evento 3

**Parámetros de entrada**:

- $ruta: Directorio de usuario de sistema "/home/[nombre_de_usuario_de_sistema] /principal" en la máquina cliente, es decir, el directorio raíz de trabajo.
- $archivo: Nombre del archivo de texto 'Resultados.txt', donde se guardan los

resultados finales de la ejecución del módulo Engine.

- $rutaObjetivo: Ubicación del archivo con nombre $archivo en la máquina cliente.
- @datosArchivo: Esta variable se corresponde con la expresión Perl "`split(/:/,$archivo);`", para obtener la IP de una máquina remota.
- @IP: Array con los cuatro números que forman parte de una dirección IPv4, en este caso la del servidor que ejecuta la última subtarea. Esta variable se corresponde con la expresión Perl "`split(/./,$datosArchivo)`";

**Descripción**:

Invocada dentro del bucle principal del script 'tareas.pl', en este archivo log se registra los últimos eventos de la ejecución, cuando el monitor verifica que se cumplen las condiciones para detener la ejecución del módulo Engine (cuando se ha encontrado solución al problema de factorización). El archivo de resultados globales de ejecución $archivo, se termina de completar en $rutaObjetivo tras finalizarse la última tarea en la máquina $IP. Se inserta la etiqueta '</file_events>', que indica el fin del archivo log XML.

#### 5.3.2.4 Configuración final y ejecución de Codeswarm

El objetivo de la representación visual de la ejecución del módulo Engine con Codeswarm consiste en mostrar con el mayor nivel de descripción posible todas las creaciones, modificaciones y transferencias de archivos entre máquinas. Para ello se optó por una configuración en la que:

- Las cadenas de caracteres identifican la máquina cliente con la palabra 'Cliente' y a los servidores con sus direcciones IP.

- Dichos identificadores de máquina son inmóviles, para evitar la tendencia a la superposición continua del programa debido a las fuerzas de atracción que generan la gran cantidad de archivos que aparecen en las secuencias, con el fin de conseguir un resultado más legible y poder rastrear las transferencias de archivos.

- Los puntos luminosos en la órbita de los identificadores se mantienen durante el mayor tiempo posible en pantalla, aunque limitando este tiempo para evitar que se genere una gran confusión con un número muy elevado de puntos que no "entren" todos dentro de la órbita y salgan despedidos demasiado pronto.

- Si se tiene en cuenta el punto anterior, tras un determinado tiempo los puntos luminosos se distancian de la órbita y se van desvaneciendo hasta desaparecer.

- Dado el gran número de ficheros '.txt', se ha configurado su asignación de colores de tal modo que en un principio sean blancos, para pasar a ser de otro

cuando sus puntos luminosos correspondientes se distancia de la órbita en la que se encontraban.

- Los puntos luminosos que representan archivos que aumentan gradualmente de tamaño o que son referenciados con un ritmo constante incrementan poco a poco su tamaño, adquiriendo grandes dimensiones.

- La estética general que se pretende reflejar puede parecerse a las instantáneas de las revistas científicas que ilustran fenómenos del espacio exterior.

**Pesos y colores de los diferentes tipos de archivo**

Éstos son los valores numéricos de la etiqueta 'weight' y los colores atribuidos a los archivos según su extensión, importancia o relevancia dentro de la ejecución global o su tamaño:

| TIPO DE ARCHIVO | Relevancia / Peso | Color |
|---|---|---|
| **Archivos de texto** | 75 (infoExec)<br>100 (resultados) | Blanco |
| **Claves públicas** | 75 | Violeta |
| **Scripts Perl** | 50 | Azul claro |
| **Sin extensión / ejecutables** | 100 (ejecutable de algoritmo)<br>50 (lista de clave autorizada) | Amarillo |
| **Archivo XML** | 75 | Rojo |

Tabla 5.1: Valores numéricos para la etiqueta 'weight' y su correspondiente código de colores.

Se ha querido dar mayor importancia a los archivos de texto con los resultados de la ejecución de las subtareas y los algoritmos ejecutables de división por tentativa y la criba cuadrática. Como se podrá comprobar tras ver un video generado por Codeswarm, cobran mayor protagonismo la máquina cliente y los servidores con mayor capacidad de cómputo, ya que son las computadoras que gestionan un número más grande de ficheros.

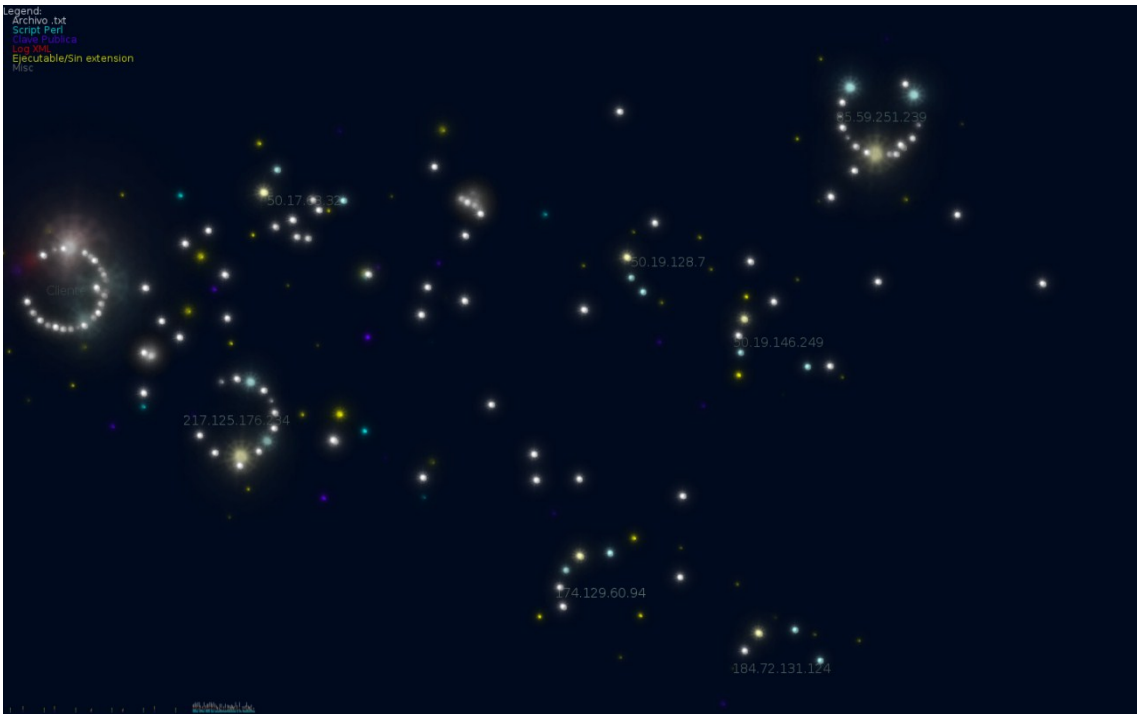


Figura 5.18: Captura de Codeswarm. Log creado en tiempo real con una ejecución del módulo Engine. Esta captura representa una fase temprana tras el arranque del sistema en la cual, después de realizar las conexiones pertinentes, la máquina cliente comienza a mandar subtareas a la vez que los servidores las ejecutan.

**Instrucciones para compilar y ejecutar Codeswarm v0.1**

- sudo apt-get install ant // instalar ant
- sudo apt-get install sun-java6-jdk // instalar JDK
- cd [ruta_carpeta_Codeswarm]
- ant // módulo compilado
- ant run // ejecutar Codeswarm con repositorio online
- sh run.sh // ejecutar Codeswarm con fichero log XML

**Instrucciones para construir un video a partir de imágenes**

- Antes de la ejecución, configurar 'simple.config':
TakeSnapshots=true SnapshotLocation=[ruta_deseada]/code_swarm-#####.png
-> Instalar un creador de video (mencoder o ffmpeg):
sudo apt-get install mencoder/ffmpeg
-> Generar video a 24 frames por segundo:
mencoder mf://*.png -mf fps=24:type=png -ovc lavc -oac copy -o movie.avi
-> O bien creando un video conservando la calidad de imagen:
ffmpeg-f image2 -r 24-i ./frames/code_swarm-%05d.png -sameq ./out.mov -pass2

## 5.3.3 Forecaster

### 5.3.3.1 Introducción: motivaciones

El primer módulo consta de los elementos necesarios para un sistema que permite la ejecución paralela de los algoritmos de factorización implementados para decodificar claves RSA en la Nube, pero sirve de muy poco si no se procede antes a una estimación relativamente aproximada de qué medios se necesitan para hacer ese cálculo (tipo y número de máquinas), cuánto tiempo tardará y la cuantía de la inversión económica que se requiere a la hora de alquilar la infraestructura necesaria.

Toda investigación o experimento, necesita antes de una predicción sobre el papel de cuál será el resultado final aproximado, para luego contrastar los cálculos teóricos con la práctica, en gran medida porque los medios económicos y el tiempo de los que se dispone es limitado. Por tanto, la iniciativa de implementar un segundo módulo surge con la idea de crear una herramienta que ayude a orientar al usuario y esbozar un cálculo o estimación aproximada de cómo se desarrollará la ejecución de factorizar una clave en concreto mediante el Engine en términos de coste y tiempo empleados en función de la configuración empleada.

Los cálculos teóricos mencionados anteriormente, en nuestro caso, serán efectuados por una aplicación con una interfaz gráfica y sencilla que permita definir cuál es la capacidad de un algoritmo dado para resolver un problema de un tamaño determinado y en unas circunstancias concretas (recursos físicos empleados y tamaño de las subtareas), y de este modo comprobar su viabilidad en la práctica sin coste ninguno y en poco tiempo. Es muy importante que se puedan definir diferentes modos de ejecución y opciones en función de qué condiciones (o parámetros dentro del programa) son fijas para cada caso y cuáles se dejan al propio módulo para obtener unos resultados más provechosos con el fin de obtener como salida la variable que queremos despejar, como si de una ecuación matemática se tratara. En ocasiones querremos saber cuánto tiempo y coste conlleva la terea, pero en otras podremos fijar alguno de éstos parámetros para averiguar los recursos necesarios o fijar éstos últimos o ninguno.

Para terminar, también sería muy conveniente, y de hecho puede que sea el aspecto más importante y útil de este módulo que el sistema pudiese orientar al usuario del módulo Engine de cuál es la configuración más óptima en función del concepto de Coste/Rendimiento (C/R) de entre todas las posibles (dentro de las limitaciones de cada caso según sus necesidades o preferencias) que otorgue un mejor aprovechamiento al dinero invertido.

### 5.3.3.2 Herramientas

Para decidir qué herramientas son las más adecuadas a la hora de implementar este módulo, debemos de tener en cuenta principalmente que lo más importante es desarrollar una interfaz gráfica sencilla que facilite al usuario una solución rápida a la hora de realizar estimaciones. Es por ello que nos decantamos por emplear Java, mediante la herramienta de desarrollo para este lenguaje.

Este segundo módulo está escrito en Java por la comodidad que facilita este lenguaje a la hora de programar rápidamente, ya que teníamos total libertad de elección al no estar obligados a ninguna tecnología en concreto. Además, Java es un lenguaje que permite total portabilidad; es totalmente independiente del sistema operativo. Por otra parte, Java ofrece muchas facilidades a la hora de crear interfaces gráficas gracias a las librerías de clases y funciones de componentes gráficos 'java.awt'[43] y 'java.swing'[44]. Con estas librerías es sencillo crear *"frames*" interactivos con paneles, botones, menús de elección, etiquetas, áreas de texto, barras de desplazamiento y demás componentes habituales en una interfaz gráfica eficaz. A estos componentes se les puede aplicar ciertas normas para posicionarlos en la interfaz, se trata de los *"layouts"*[45], un tipo de clases predefinidas con las que podemos definir la ubicación y ordenación dentro de los diferentes paneles y *"frames"* a nuestro antojo, pudiendo habilitar márgenes, celdas invisibles, posiciones relativas para cada componente, etc.

Otra ventaja de Java es que es un lenguaje tan extendido que se pueden encontrar innumerables librerías de código abierto para cualquier propósito con el fin de añadir módulos al proyecto que se esté desarrollando para ahorrar tiempo en su implementación y simplificar la arquitectura del mismo. De este modo, un programador puede aprovechar un módulo que otro programador ha implementado anteriormente, agilizando así el proceso de desarrollo de un proyecto. Así mismo, Oracle proporciona una extensa serie de material de documentación y tutoriales, que son muy útiles a la hora de buscar formas de implementación que se sean efectivas. Además, en la Red existen también incontables foros de discusión exclusivos de esta plataforma en los que los internautas tratan de darse soporte mutuamente en la resolución de problemas. El único requisito necesario para desarrollar y ejecutar programas en Java es haber instalado anteriormente una máquina virtual de java compatible con el sistema operativo de la máquina física.

Posteriormente, para crear un programa en java se tuvo que escoger un entorno de desarrollo como las plataformas NetBeans o Eclipse. En el caso de este proyecto, se optó por la primera, esencialmente por una sencilla razón: a la hora de desarrollar ciertas ventanas o paneles de cierta complejidad, esta herramienta proporciona una útil interfaz para crear GUIs rápidamente. Este editor ha sido empleado para crear paneles con un número considerable de botones, etiquetas y otros elementos en los que muchos condicionantes entraban en juego a la hora de crear una interfaz que se adaptara a todos los tipos de resolución de monitor posibles.

[43] http://download.oracle.com/javase/1.4.2/docs/api/java/awt/package-summary.html
[44] http://download.oracle.com/javase/1.4.2/docs/api/javax/swing/package-summary.html
[45] http://download.oracle.com/javase/tutorial/uiswing/layout/visual.html

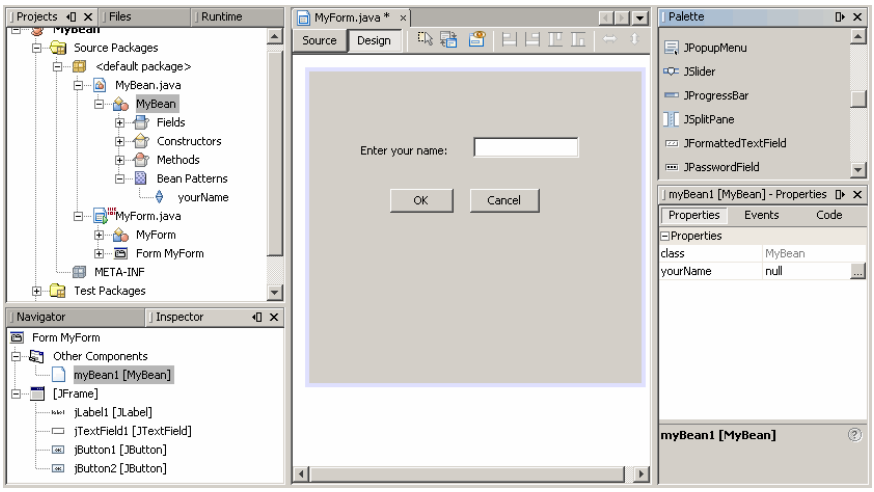


Figura 5.19: Editor de GUIs de Netbeans.

Cabe destacar que, con el objetivo de ayudarnos a implementar gráficas de funciones matemáticas, se optó por buscar alguna librería que hiciese más fácil nuestro trabajo. Es por ello que el módulo Forecaster incluye un conjunto de librerías de código abierto cuyo propósito es facilitar la creación de un componente que represente gráficamente funciones matemáticas con una interfaz cómoda y fácil de usar: JCM (*"Java Components for Mathematics version 1.0"*) del Departamento de Matemáticas e Informática de Hobart and William Smith Colleges[46]. Este proyecto representa un esfuerzo para desarrollar un "*framework*" de componentes de software matemático escrito en el lenguaje de programación Java destinado a utilizarse para la construcción de programas con "*applets*" e ilustraciones interactivas de funciones, analizadores sintácticos o "*parser*" matemáticos, calculadoras de propósito especial para operandos grandes, etc. Los componentes de la versión 1.0 son en su mayoría útiles para el cálculo y pre-cálculo y están orientados para el uso académico y de investigación. Todos ellos emplean Java 1.1, por lo que no funciona en algunos navegadores antiguos que sólo admiten Java 1.0 en el caso de realizar una página web. También son utilizables mediante un programa Java que se ejecute en una máquina virtual JDK, como es nuestro caso. Incluir las clases que se requerían de esta librería es tan sencillo como hacer doble click, gracias a editor Netbeans.

Importante ha sido el uso de la herramienta MatLab[47], necesaria para el cálculo previo de las funciones de predicción que posteriormente se implementaron en Java. MatLab es un software matemático que ofrece un entorno de desarrollo integrado (IDE) con un lenguaje de programación propio. Se ha usado esta herramienta para la gran mayoría de los cálculos por su gran capacidad de cálculo y por ofrecer una mayor precisión frente a otras herramientas.

Por último, para la ejecución de este módulo adoptamos la decisión de crear un archivo ejecutable con extensión '.jar' propios de la máquina virtual de Java que hiciese más rápido y sencillo su manejo para cualquier usuario, sin importar su nivel de conocimientos. Es sin duda la vía más rápida pero no por eso deja de ser efectiva.

[46] http://math.hws.edu/javamath/
[47] http://www.mathworks.es/

#### 5.3.3.3 Definición de funciones y cálculos previos

Para llevar a cabo la estimación del tiempo y presupuesto se requiere cierta tarea representativa. Inicialmente llevamos a cabo el estudio del comportamiento de los algoritmos en cada tipo de máquina de Amazon. Para ello, se escogen distintos tamaños de clave, se ejecuta el algoritmo, y se obtiene tiempo total resultado de la ejecución de dicho algoritmo.

Para el caso del algoritmo división por tentativa (ver sección 3.6.1) se ha ejecutado el algoritmo sobre cuatro tipos de máquinas diferentes (Small, Large, High CPU Medium, High CPU ExtraLarge), obteniendo los siguientes resultados.

| Clave | Tiempo Small Horas | Tiempo Large Horas | Tiempo High CPU Medium Horas | Tiempo High CPU ExtraLarge Horas |
|---|---|---|---|---|
| 1037239934749514921 | 0,0275425 | 0,013549 | 0,0114 | 0,013721 |
| 108582723003788360651 | 0,3711 | 0,1369 | 0,16228639 | 0,137328 |
| 10875662676677624740093 | 4,371803333 | 1,38857 | 1,915 | 1,392659 |
| 1012994046101375366298691 | 41,37904056 | 13,41 | 17,61715139 | 14,130443 |

Tabla 5.2: Resultados obtenidos al ejecutar una tarea con el algoritmo de división por tentativa en distintas máquinas de Amazon.

A partir de los resultados, hemos de obtener una recta de regresión que explique el comportamiento de la ejecución del algoritmo, en función del tiempo, para distintos tipos de instancias. Para calcular la regresión que mejor se ajuste a los datos obtenidos, se hizo uso de la herramienta MatLab, implementando un código, que realiza distintos ajustes de regresión por mínimos cuadrados. Los ajustes son exponencial, potencial, logarítmico y polinomial. Finalmente, puede visualizarse en pantalla la ecuación del modelo ajustado y su gráfica correspondiente. Para todos los tipos de instancias se obtuvo un mejor ajuste con la regresión potencial, obteniendo así la ecuación que mejor define su comportamiento. En la siguiente Figura 5.20 se puede ver la regresión con ajuste potencial para una instancia.

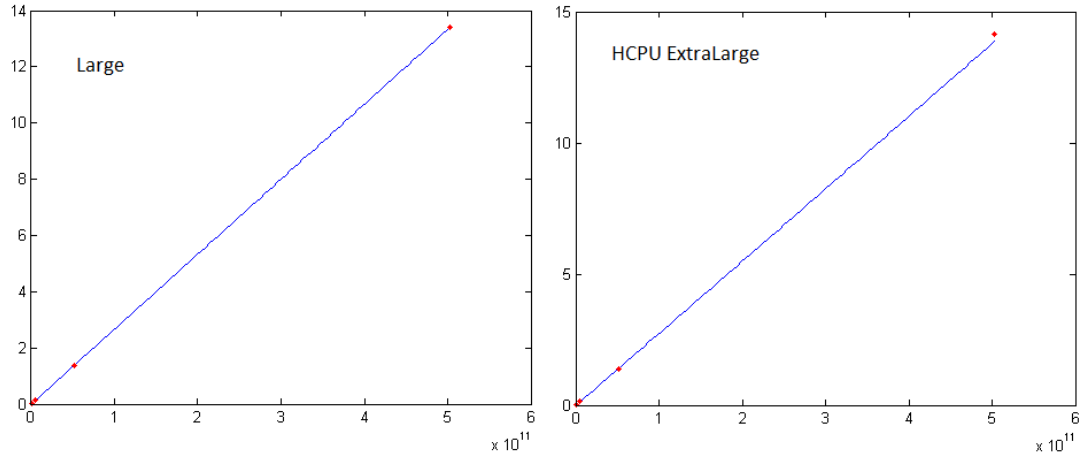


Figura 5.20: Gráfica de regresiones Large y HCPU ExtraLarge

Una vez definida la recta de regresión que describe el comportamiento del algoritmo en cada tipo de instancia para tareas individuales, se puede generalizar la ecuación del tiempo total T para el modelo de paralelización resultante de dividir el trabajo de factorizar cierta clave en un determinado número de subtareas mediante la siguiente ecuación [VAPO]:

$$T = \frac{t_{exec(i)}\, I}{i\, N_{maqs}\, N_{núcleos}} \qquad (Ec. 5.1)$$

Donde $t_{exec(i)}$ es la función resultante de las regresiones obtenidas anteriormente, *'I'* e *'i'* son el intervalo total y el intervalo procesado en cada subtarea respectivamente, $N_{maqs}$ es el número de máquinas virtuales instanciadas en el experimento y $N_{núcleos}$ el número de núcleos de cada instancia.

El valor de *'I'* está determinado por el tamaño de la clave, cuyo valor es igual a

$$I = {\sqrt{clave}}/{2} \qquad (Ec. 5.2)$$

Para estas estimaciones, el tamaño de cada tarea *'i'* ha sido:

$$i = \frac{I}{kN_{maqs}N_{núcleos}} \qquad (Ec. 5.3)$$

Donde *'k'* es el número de tareas asignadas a cada núcleo de cada máquina.

A la hora de elegir una configuración óptima, el tiempo total de ejecución no es la única condición a tener en cuenta. Es necesario establecer una relación entre el tiempo T y su coste asociado, cuyo valor resultante es:

$$C = C_{hora}N_{maqs}\lceil T \rceil \qquad (Ec. 5.4)$$

Donde $C_{hora}$ el coste del uso del tipo de instancia elegida por hora tal y como se describe en la Tabla 5.2. A la variable *'T'* se le aplica la función techo debido a que los precios corresponden a cada hora de uso de la máquina solicitada.

La configuración óptima está determinada por la búsqueda de un compromiso entre el tiempo y el coste denominado Coste/Rendimiento (C/R) .Esta relación se obtiene al multiplicar ambos parámetros y la configuración más conveniente corresponde a su valor mínimo:

$$C/R_{óptimo} = min(C/R) = min(CT) = mi\,n\left(\frac{C_{hora}t_{exec(i)I}}{iN_{núcleos}}\left\lceil\frac{t_{exec(i)I}}{iN_{maqs}N_{núcleos}}\right\rceil\right) \quad (Ec.\,5.5)$$

Donde las variables corresponden a las fórmulas Ec 5.1 y Ec. 5.4 y a la instancia seleccionada. El valor óptimo de C/R se alcanza cuando el número de instancias utilizado hace que el tiempo de uso de cada máquina instanciada sea exactamente de una hora. Tras obtener el mejor C/R para cada máquina, podemos evaluar cuál de ellas es mejor para el problema abordado y averiguar el número de máquinas virtuales necesarias para el trabajo total en el mejor caso con la expresión anterior.

#### 5.3.3.4 Descripción: modos de ejecución

Profundizando un poco más, podemos decir que el Forecaster es una herramienta muy útil para calcular una estimación de cuantas máquinas de Amazon EC2 se deben contratar y/o durante cuánto tiempo para realizar una tarea de factorización determinada en función de uno de los siguientes parámetros: tiempo, número y tipo de máquinas disponibles, presupuesto disponible, tamaño de cada tarea independiente o la mejor relación Coste/Rendimiento.

Inicialmente dispone al usuario de cinco tipos de predicción que determinan cuál es la entrada o "*input*" de datos y por tanto qué salida "*output*" espera del módulo para dar respuesta al problema de estimación planteado. Tras establecer las preferencias y parámetros iniciales, el programa nos muestra un gráfico que determina la relación entre la clave y el tiempo necesario para obtener sus factores y un informe con los resultados en función de los datos de entrada.

**Descripción de los diferentes modos de ejecución**

Para entender mejor el funcionamiento de este módulo, será apropiado dar una explicación de los cinco tipos de estimación de los que dispone. En primer lugar, nada más ejecutarse el módulo contenido en un archivo ejecutable '.jar', se despliega una ventana con un menú en el que se enumeran y detallan todas las modalidades. En la parte izquierda aparece un grupo de cinco botones Radio Button con el con el nombre de cada modo de ejecución seguidos de un botón 'Aceptar' y en la derecha hay un recuadro de texto en el que al seleccionar sobre alguna de la opciones disponibles, podemos leer una breve descripción de la misma.

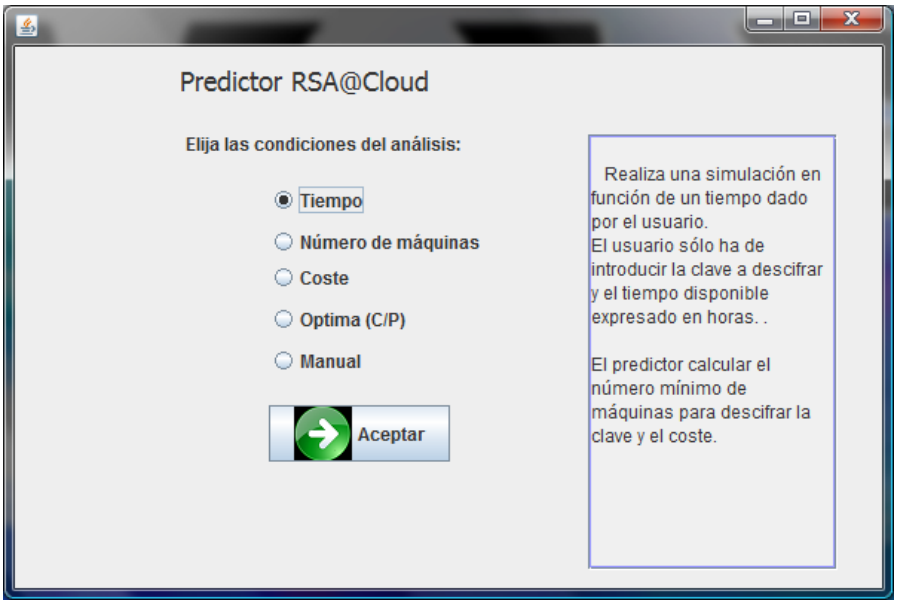


Figura 5.21: Panel inicial.

Los diferentes modos de ejecución del Forecaster en función de las condiciones del análisis de estimación deseado son:

- Tiempo
- Número de máquinas
- Coste
- Óptima
- Manual

En todos ellos, tras escoger un modo y pulsar el botón 'Aceptar', se pasará a la ventana principal del programa, donde podemos encontrar un panel a la izquierda en el cual se introducen los datos de entrada, un panel superior derecho donde se muestra una gráfica comparativa que muestra las funciones de los tiempos de ejecución en función del tamaño de la tarea (o intervalo *'i'*) de cada uno de los cuatro tipos de instancias de Amazon EC2 empleadas y un panel derecho inferior donde se muestran los resultados del análisis requeridos.

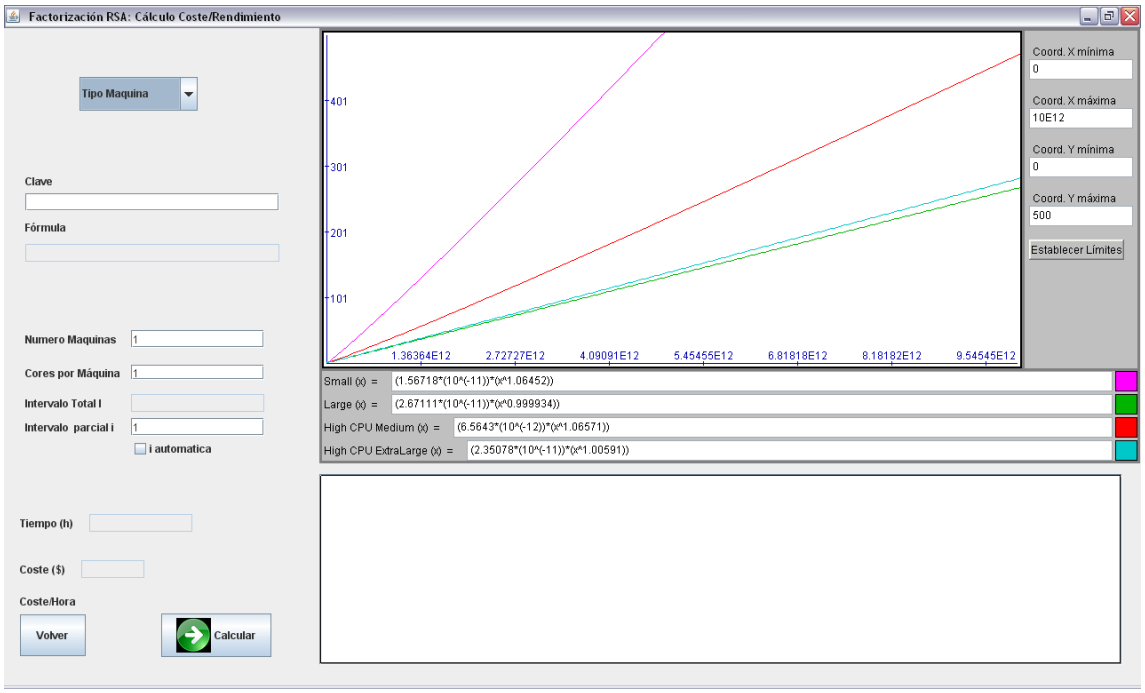


Figura 5.22: Ventana principal.

### Descripción de la ventana principal

Tras elegir la opción deseada, una ventana muestra tres paneles: los datos de entrada, una gráfica comparativa entre los distintos tipos de instancia y un panel donde se mostrarán los resultados finales:

- Panel de entrada de datos (a la izquierda): En cada uno de los modos, aparecerá siempre en la parte superior del panel izquierdo un ComboBox llamado 'Tipo Máquina'. Para proceder al cálculo de una estimación con el botón 'Calcular', todos los parámetros editables, incluido éste, deberán tener algún valor (en caso contrario, aparecerán mensajes de advertencia informando al usuario que debe completar todos los campos editables). Se puede elegir el tipo de instancia que usarán las máquinas virtuales destinadas a ejecutar el algoritmo de factorización. Existen las cinco opciones diferentes siguientes: 'Small', 'Large', 'High CPU Medium', 'High CPU Large' y 'Máquina Óptima' En esta última opción, el sistema escogerá la máquina que acarree el menor coste económico necesario para ejecutar el cálculo. Los otros campos, editables o no, en función del modo de ejecución escogido son: 'Clave', 'Fórmula' (de la instancia seleccionada), 'Número de Máquinas', 'Número de Cores' (por máquina), 'intervalo I' (o tamaño total de la tarea), 'intervalo i' (o tamaño de cada subtarea), 'i automática' (un CheckBox que habilita un tamaño de intervalo i óptimo), 'Tiempo' y 'Coste' (en dólares americanos). También está disponible un campo de tipo Label llamado 'Coste/Hora' (en dólares americanos) que depende de la instancia elegida. Por último, además del botón "Calcular" existe otro llamado 'Volver' para regresar al panel inicial.

- Panel gráfico (arriba a la derecha): En este panel se muestra la representación gráfica de las funciones de tiempo de ejecución en horas de las distintas instancias con respecto al tamaño de la tarea, o dicho de otro modo, el tamaño del intervalo *'i'*. Estas funciones expresan la relación entre el número de factores que se deben probar para una clave determinada y el número de horas de ejecución. Además, en la parte de la derecha existen cuatro campos editables y un botón llamado 'Establecer Límites' con los que se pueden definir los márgenes superiores e inferiores de las coordenadas X e Y en representación de la gráfica. En la parte inferior se despliega una leyenda en la que se relacionan los tipos de instancia con la función de tiempo característica y su color correspondiente en el gráfico.

- Panel de resultados (abajo a la derecha): Inicialmente vacío cuando aparece la ventana principal, muestra los resultados del análisis tras completar todos los campos editables y pulsar el botón 'Calcular'. La información mostrada da valores para los siguientes parámetros: tipo de máquina, número de cores por máquina, clave, intervalo total *'I'*, intervalo por tarea *'i'*, número de máquinas, tareas por core, tiempo por cada tarea (en horas, minutos y segundos), tiempo total (en horas, minutos y segundos), coste por hora (en dólares americanos), coste (en dólares americanos) y relación Coste/Rendimiento.

A continuación se describen los diferentes modos de ejecución:

**Tiempo**

En esta modalidad, el usuario establece un tiempo en horas máximo exigido para solucionar el problema de factorización de una clave que él mismo introduce. Como salida, se muestra el número de máquinas virtuales mínimo y el coste que implica la ejecución del algoritmo en los límites de tiempo impuestos.

Una vez se ha elegido esta opción, aparece la ventana principal. En el panel izquierdo se muestran como editables los siguientes parámetros: el tipo de máquina, la clave y el tiempo de ejecución.

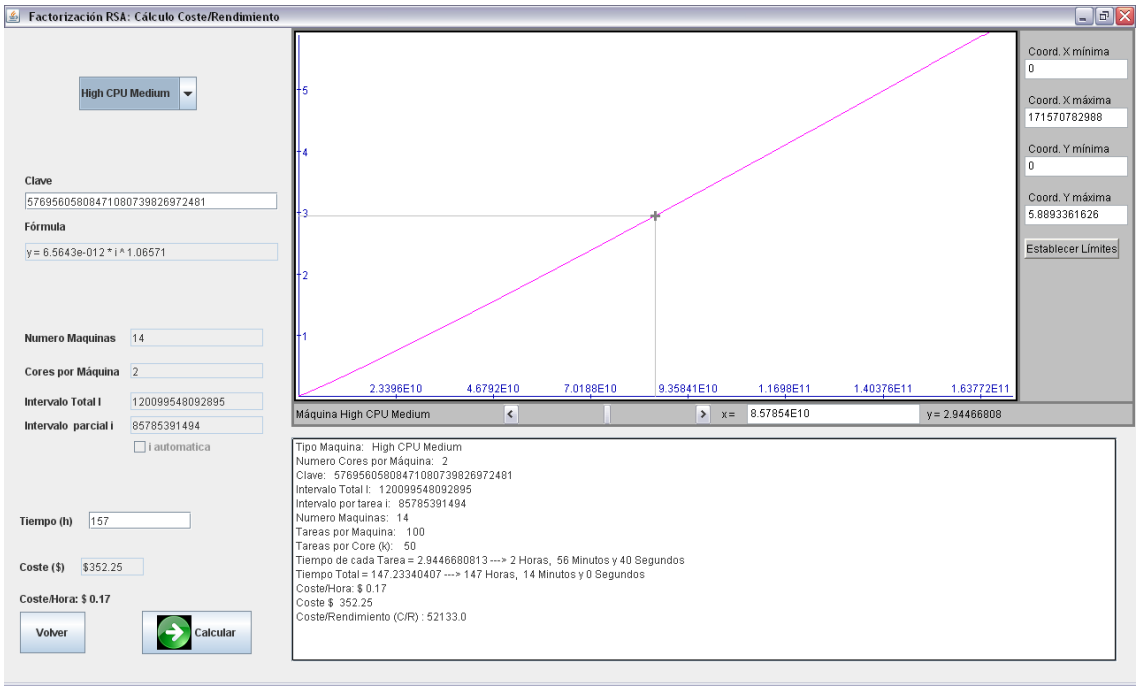


Figura 5.23: Modo Tiempo.

Para obtener el número mínimo de máquinas virtuales necesarias para descifrar la clave en el tiempo establecido por el usuario, inicializamos el tiempo límite al tiempo fijado y el número de máquinas a uno, de tal forma que a medida que avanza la búsqueda éste se va incrementando. El algoritmo va calculando el tiempo para cierto número de máquinas, de forma que si el tiempo obtenido es mayor que el tiempo límite, incrementa en uno el número de máquinas, hasta que el tiempo sea menor que el tiempo establecido. Por último se calcula el coste y se muestran los resultados.

```
PREDICCIÓN TIEMPO
    Obtenemos el tiempo límite introducido por el usuario.
          tiempoLimite = this.tiempo;
    Fijamos el tiempo total al valor máximo.
          this.tiempo = MAX_VALUE;
    Iniciamos el número de máquinas a 1.
          this.numMaq = 1;
    Buscamos la mejor relación tiempo/coste, sin superar el tiempo limite.
    MIENTRAS (tiempoLimite <= this.tiempo) HACER
           calculaTiempo();
           this.numMaq ++;
    FIN MIENTRAS
    Calculamos el coste final.
    Mostramos los resultados.
FIN PREDICCIÓN TIEMPO
```

Figura 5.24: Algoritmo de la predicción Tiempo.

### Número de máquinas

Esta opción permite fijar el número de máquinas disponibles para la ejecución de la tarea de factorizar el número introducido. La salida obtenida es el tiempo y precio necesarios para la clave dada con dicho número de máquinas.

Una vez se ha elegido esta opción, aparece la ventana principal. En el panel izquierdo aparecen como editables los siguientes parámetros: el tipo de máquina, la clave y número de máquinas.

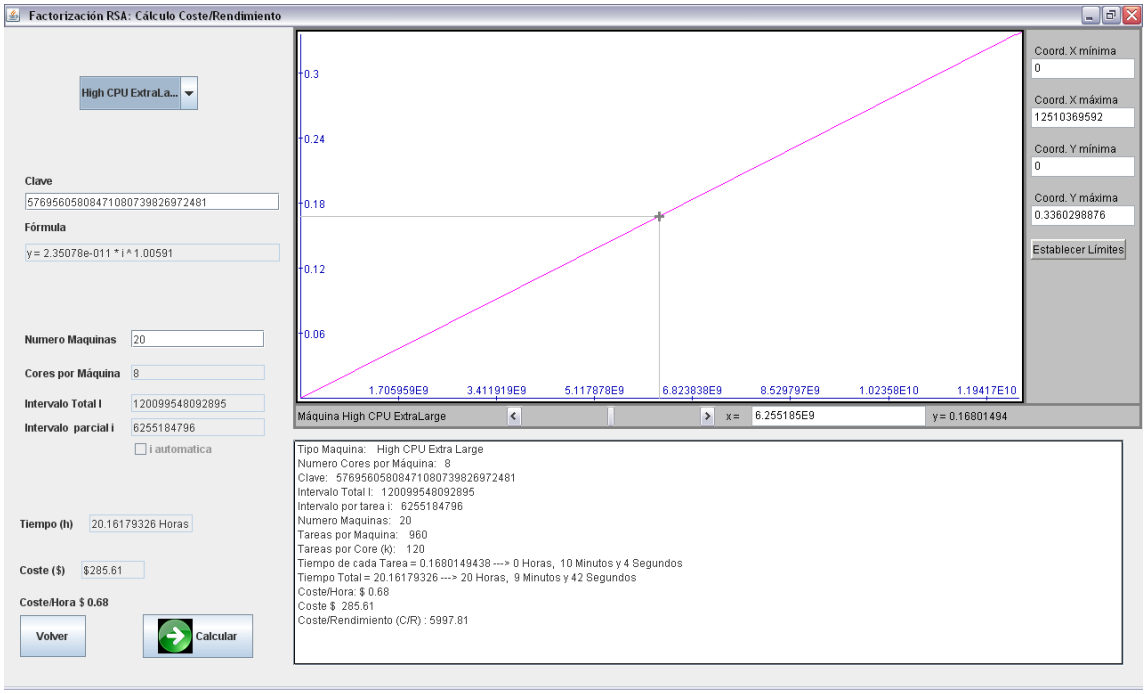


Figura 5.25: Modo Número de máquinas.

Cuando se selecciona esta modalidad, sólo hay que calcular el tiempo invocando la función propia del tipo de instancia seleccionada, calcular el coste y mostrar los resultados.

```
PREDICCIÓN MÁQUINAS
    Calculamos el tiempo.
          calculaTiempo();
    Calculamos el coste final.
          CalculaCoste();
    Mostramos los resultados.
FIN PREDICCIÓN MÁQUINAS
```

Figura 5.26: Algoritmo de la predicción Máquinas.

**Coste**

Con esta configuración, el usuario introduce el presupuesto máximo disponible para descifrar cierta clave. Esta estrategia estima el número mínimo de máquinas necesarias que resolvieran la tarea y el tiempo empleado para ello. El máximo de máquinas que el programa establece como límite es 20, que son las instancias que Amazon EC2 permite desplegar simultáneamente sin tener que enviar un formulario de petición para ampliar dicho límite.

Una vez se ha elegido esta opción, aparece la ventana principal. En el panel izquierdo aparecen como editables los siguientes parámetros: el tipo de máquina, la

clave y coste. La salida resultante indicará si es posible o no realizar la tarea correspondiente con ese presupuesto. En caso afirmativo, señalará el número de máquinas necesario y el dinero sobrante, en caso contrario se informará de ello y se ofrecerán los resultados para el caso en el que se hubiera introducido el presupuesto mínimo necesario para calcular la factorización de la clave introducida.

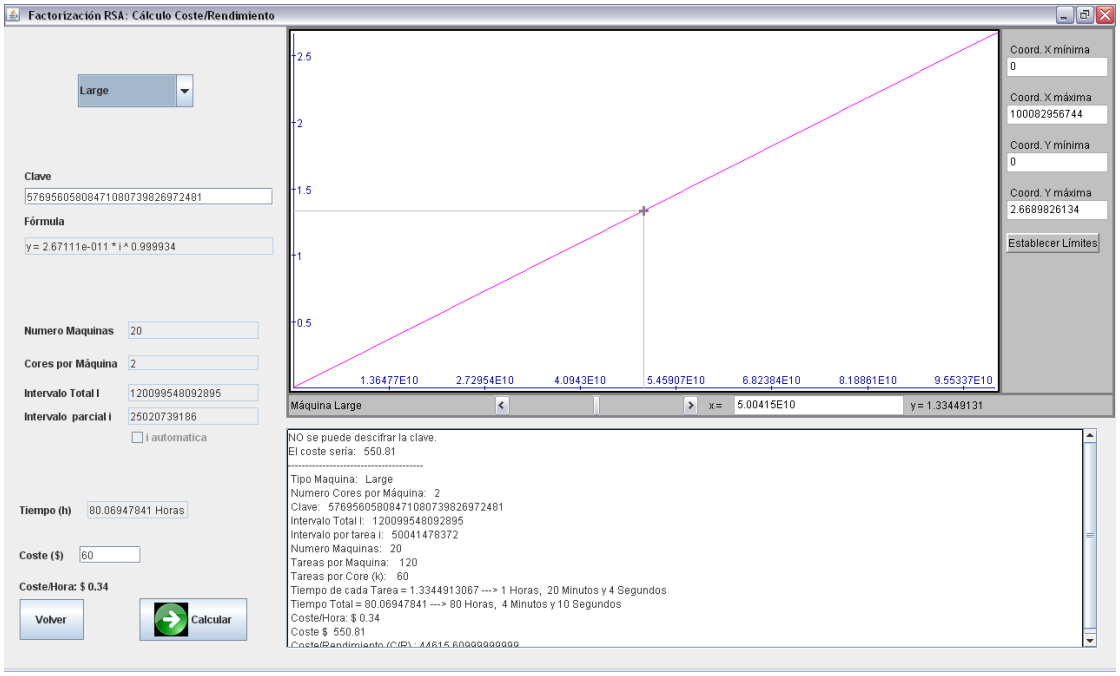


Figura 5.27: Modo Coste.

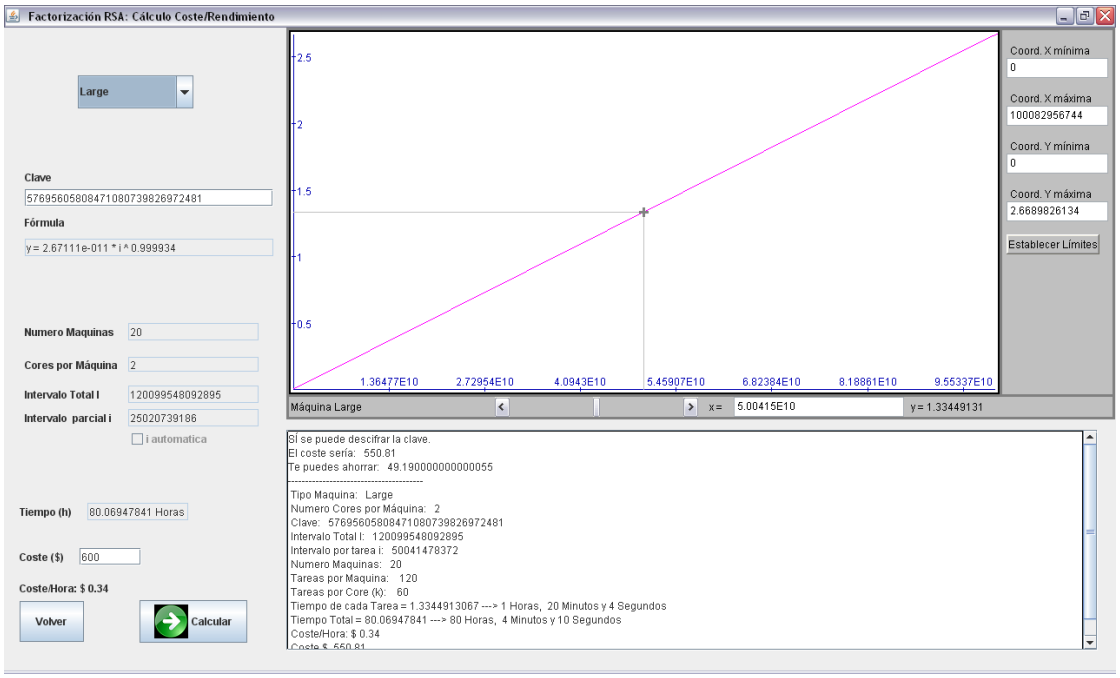


Figura 5.28: Modo Coste: presupuesto insuficiente para la clave a factorizar.

En primer lugar se calcula el número de máquinas disponibles, de forma que si es cero, se muestra el resultado de "precio insuficiente", si es mayor a veinte, se fija el número de máquinas a veinte y se obtiene el tiempo, y en caso de que sea mayor que cero y menor que veinte, buscamos la solución óptima. Finalmente se calcula el precio exacto para el tiempo encontrado y se muestran los resultados, de forma que si es un valor negativo se informa del dinero necesario para descifrar dicha clave. Por el contrario, si la diferencia es positiva se indica el dinero sobrante.

```
PREDICCIÓN COSTE
    Obtenemos el número de máquinas disponibles.
       numMaquinasMax = this.coste / this.precioHora
    CASOS
      SI numMaquinasMax = 0 ENTONCES
          «precio insuficiente»
      FIN SI
      SI numMaquinasMax >= 20 ENTONCES
         Fijamos el número de máquinas a 20.      this.numMaq = 20;
         Calculamos el tiempo para 20 máquinas.   calculaTiempo();
      FIN SI
      SINO  Búsqueda de la mejor relación tiempo/coste entre 1 y 20 máquinas
         Fijamos el tiempo máximo a 57 minutos.    tiempoLimite = 0,95;
         Iniciamos el número de máquinas a 1.      this.numMaq = 1;
         Iniciamos el tiempo total al valor máximo. this.tiempo = MAX_VALUE;
         MIENTRAS (tiempoLimite <= this.tiempo) AND (this.numMaq <= numMaquinasMax) HACER
           calculaTiempo();
           this.numMaq ++;
         FIN MIENTRAS
      FIN SINO
    FIN CASOS
    Calculamos el coste exacto para el número de máquinas y el tiempo obtenidos.
    Calculamos la diferencia con el coste inicial.
    Mostramos los resultados.
FIN PREDICCIÓN COSTE
```

Figura 5.29: Algoritmo de la predicción Coste.

### Óptima (C/R)

En esta modalidad se realiza una simulación óptima. De entre todas las configuraciones de tipo y número de máquinas posibles, elige la que tiene la mejor relación Coste/Rendimiento.

Una vez se ha elegido esta opción, aparece la ventana principal. En el panel izquierdo aparecen como editables los siguientes parámetros: el tipo de máquina y la clave. Opcionalmente, el usuario puede escoger entre editar o no el campo 'Intervalo parcial i'. Si pulsa en el Checkbox titulado 'i automática', el sistema elegirá un valor para esa variable muy conveniente en función de los cálculos teóricos y experimentos prácticos ensayados anteriormente. En caso contrario, el usuario podrá escribir el valor que mejor estime conveniente.

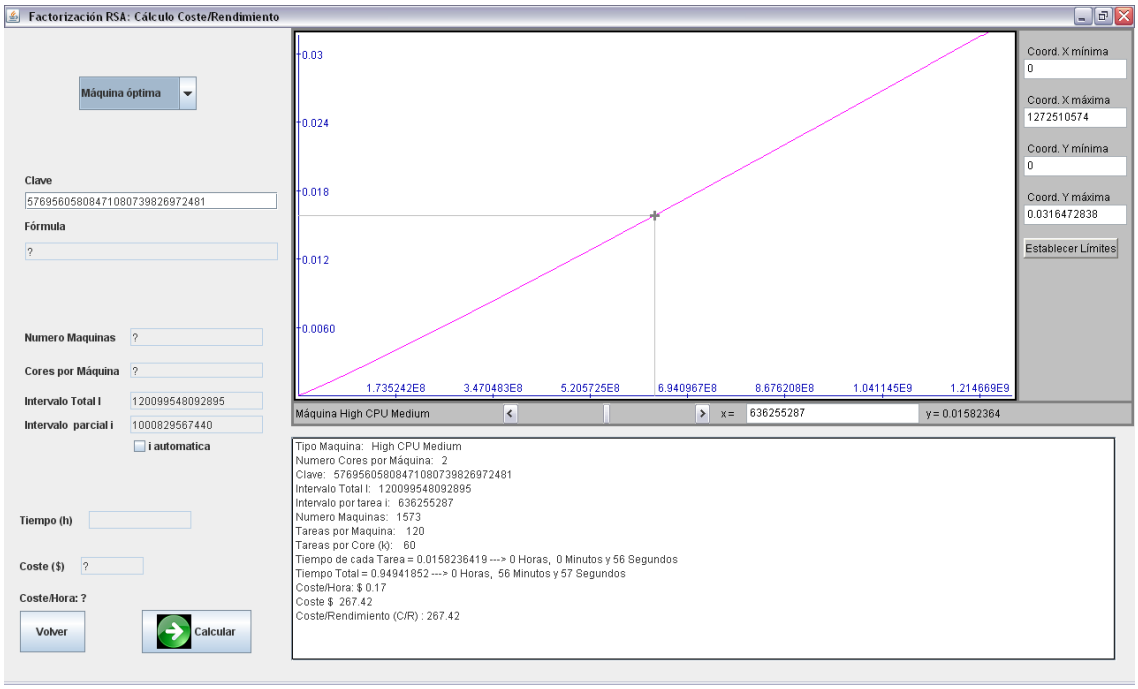


Figura 5.30: Modo Óptima (C/R).

El valor óptimo alcanza cuando el número de instancias utilizado hace que el tiempo de uso de cada máquina instanciada sea exactamente de una hora. Al igual que en la modalidad tiempo, buscamos el número de máquinas cuyo tiempo de ejecución sea inferior al tiempo límite. En este caso, como buscamos el tiempo óptimo, inicializamos el tiempo límite a 57 minutos (0.95 horas), dejando un margen de 3 minutos para transferencias de archivos y posibles retrasos. El algoritmo va incrementando en uno el número de máquinas hasta obtener un tiempo menor al tiempo óptimo. Finalmente se calcula el coste, que en este caso será óptimo, y se muestran los resultados.

```
PREDICCIÓN ÓPTIMA
    Iniciamos el tiempo límite ha 57 minutos.
          tiempoLimite = 0,95;
    Fijamos el tiempo total al valor máximo.
          this.tiempo = MAX_VALUE;
    Iniciamos el número de máquinas a 1;
          this.numMaq = 1;
    Buscamos el la mejor relación tiempo/coste, hasta que tiempo sea menor
      que tiempo limite.
    MIENTRAS (tiempoLimite <= this.tiempo) HACER
           calculaTiempo();
           this.numMaq ++;
    FIN MIENTRAS
    Calculamos el coste final;
    Mostramos los resultados;
FIN PREDICCIÓN ÓPTIMA
```

Figura 5.31: Algoritmo de predicción Óptima.

**Manual**

Como su nombre indica, en esta estrategia se da total libertad al usuario para configurar los parámetros de la predicción, incluido también el tamaño de las subtareas en la que se divide el trabajo total. La salida mostrará el tiempo y coste necesarios para completar la tarea total así como su relación Coste/Rendimiento, todo ello en función de la configuración establecida.

Una vez se ha elegido esta opción, aparece la ventana principal. En el panel izquierdo aparecen como editables los siguientes parámetros: el tipo de máquina, la clave, el número de máquinas, el número de núcleos por máquina, el intervalo total *'I'* y el intervalo parcial *'i'*.

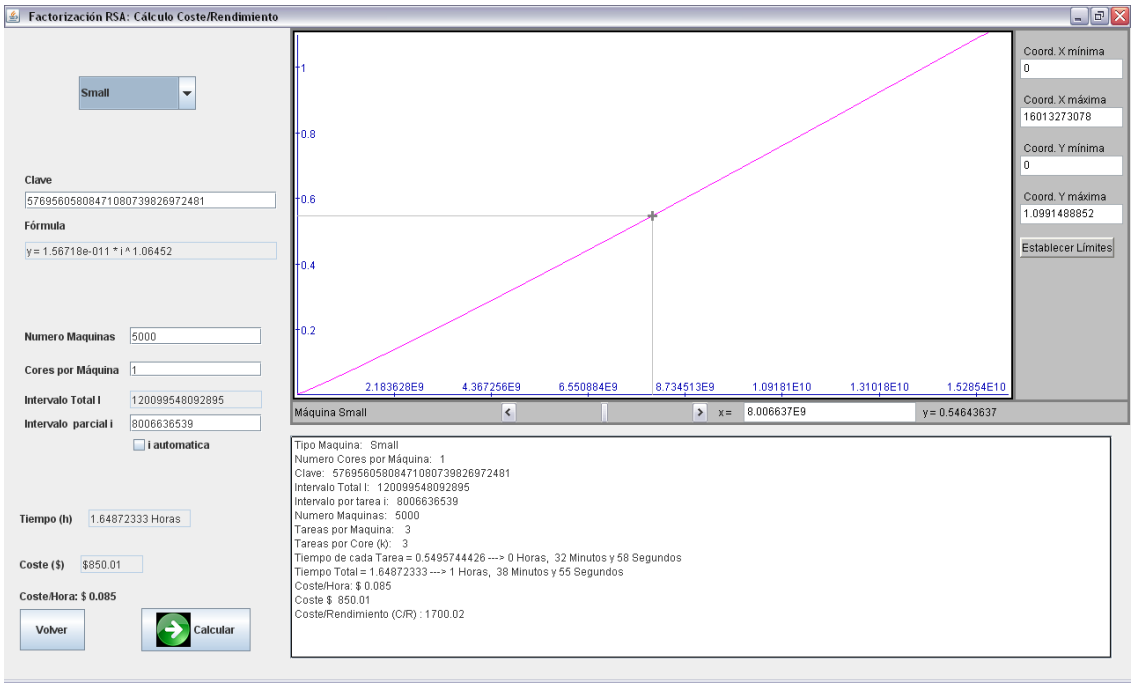


Figura 5.32: Modo Manual.

Dado que el usuario introduce todos los datos, sólo hay que aplicar la resolución matemática de la función general del tiempo (Ec. 5.1) y coste (Ec. 5.2)

## DIAGRAMAS DE CLASE

A continuación, se describe detalladamente cada paquete de diseño significativo de la arquitectura mediante una breve descripción de la funcionalidad que agrupa y su cometido, y un diagrama de clases del paquete, así como la relación del mismo con el Forecaster.

**Predictor:** Paquete que contiene todas las clases relacionadas con la interfaz gráfica. La clase 'PanelInicio' es el panel inicial en el cual el usuario elige el tipo de predicción (simulación) que más se adecue a sus preferencias. La clase 'PanelConclusiones' es un panel que muestra los resultados que se han obtenido al realizar los cálculos de una

predicción (simulación). 'PanelIzquierdo' es la clase en la cual el usuario define los parámetros necesarios para el cálculo de la predicción. 'GraficoSimple' y 'MultiGrafico' representan gráficamente la relación entre la clave y el tiempo necesario para obtener sus factores para una o varias instancias respectivamente, haciendo uso de la función de regresión obtenida para cada tipo de máquina.

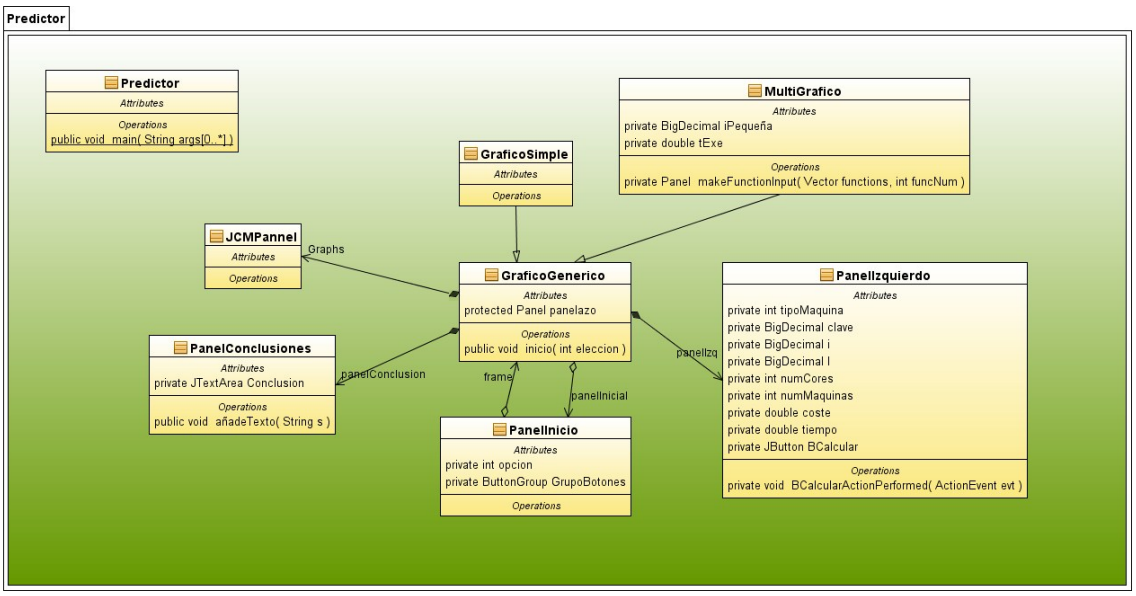


Figura 5.33: Diagrama de clases para Forecaster.

Operaciones: Paquete que contiene todas las clases necesarias para el cálculo de parámetros: tiempo, coste, número de máquinas, intervalo *'i'*, etc.... La clase 'raizBig' calcula la cantidad de factores a probar para una determinada clave, mediante la raíz cuadrada de números muy grandes mediante el Teorema de Gauss. La clase parametrizada 'Predicciones' realiza todos los cálculos necesarios para obtener el resultado de un tipo de predicción (simulación) que inicialmente ha elegido el usuario.

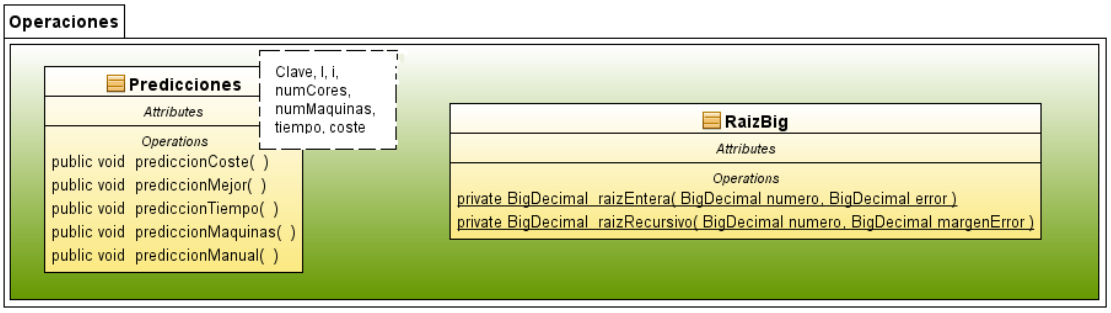


Figura 5.34: Diagrama del paquete Operaciones.

**Maquinas:** Paquete que contiene todas las clases relacionadas con el tipo de instancia. Principalmente está formado por una clase abstracta que contiene todos los tipos de atributos de cada máquina, de la que heredan todas las instancias utilizadas en el módulo Forecaster. Cada instancia tiene definida su propia función de tiempo, obtenida anteriormente mediante regresión.

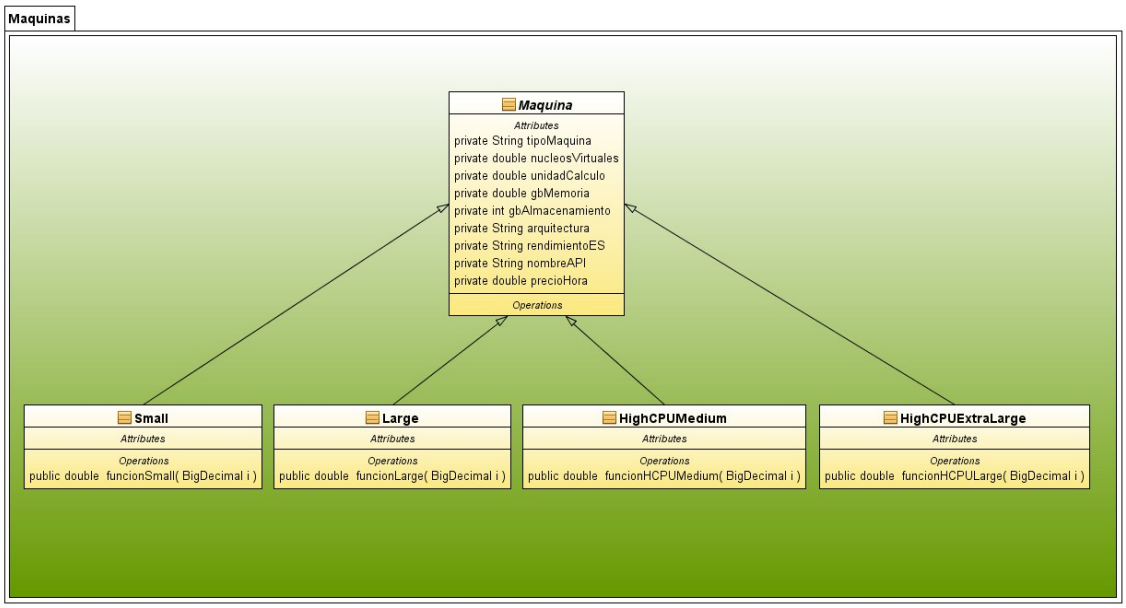


Figura 5.35: Diagrama del paquete Máquinas.

Finalmente, el diagrama completo que modela el Forecaster, obtenido a partir de la unión de los distintos paquetes:

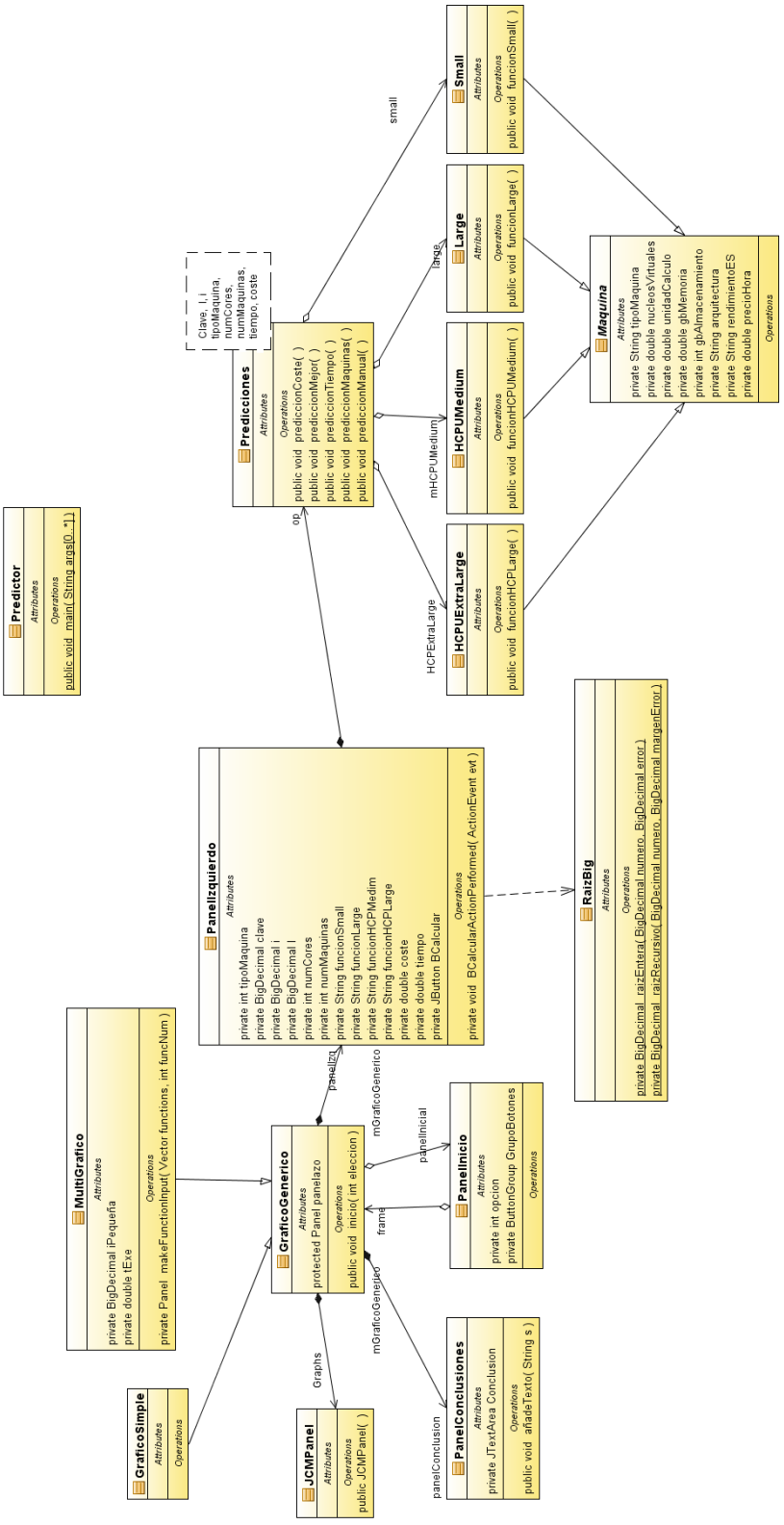


Figura 5.36: Diagrama completo del Forecaster.

## DIAGRAMAS DE ESTADO

La siguiente figura representa los diferentes estados del Forecaster, a la hora de ser utilizado por un usuario.

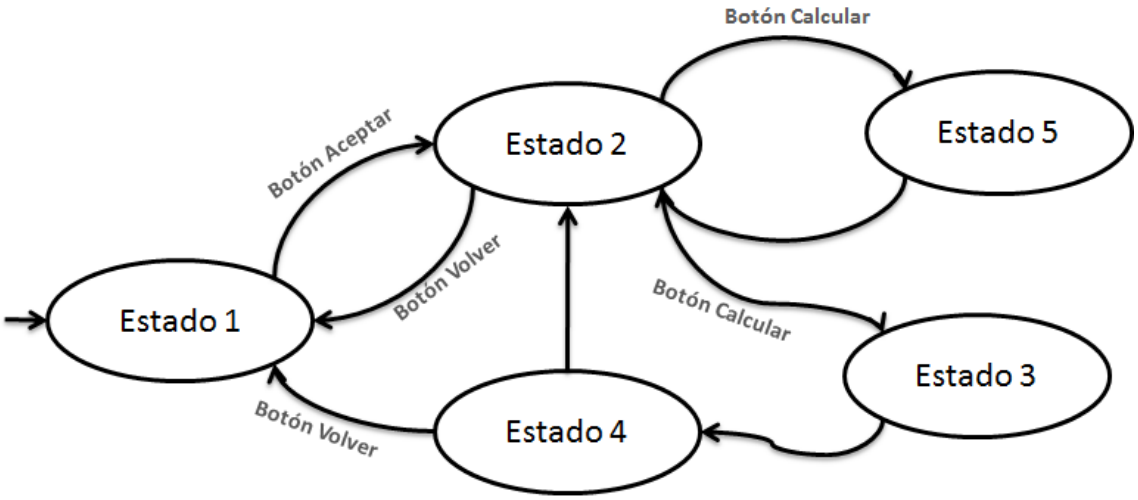


Figura 5.37: Diagrama de estados del Forecaster.

- Estado 1: Estado inicial en el cual el usuario escoge el tipo de predicción.

- Estado 2: El usuario ha de introducir los parámetros adecuados según el tipo de predicción escogida en el estado 1.

- Estado 3: Una vez pulsado el botón calcular, el Forecaster realiza los cálculos.

- Estado 4: Se muestran los resultados y conclusiones calculadas por el Forecaster. El usuario podrá volver al estado inicial en donde podrá modificar el tipo de predicción, o modificar los parámetros para un mismo tipo de predicción.

- Estado 5: El Forecaster muestra mensaje de error debido a que el usuario no ha introducido todos los parámetros requeridos para el tipo de simulación escogida.

# 6. Casos de Uso

## 6.1 Estudio y estimación: Forecaster

### 6.1.1 Resultados Experimentales

Estudio de la infraestructura óptima en el ataque a una clave: Forecaster.

Para llevar a cabo los experimentos relacionados con la estimación del tiempo y presupuesto que requiere cierta tarea representativa, primero llevamos a cabo el estudio del comportamiento del algoritmo de la división por tentativa en cada tipo de máquina de Amazon. Para ello, se escogen distintos tamaños de clave y se ejecuta el algoritmo, obteniendo la recta de regresión que explica su comportamiento. Los resultados están recogidos en la Figura 6.1:

Banco de pruebas:

- 1010189899 x 1026777179 = 1037239934749514921 (19 dígitos).
- 10240297597 x 10603473383 = 108582723003788360651 (21 dígitos).
- 103868294641 x 104706279373 = 10875662676677624740093 (23 dígitos).
- 1005680009767 x 1007272727173 =1012994046101375366298691(25 dígitos).

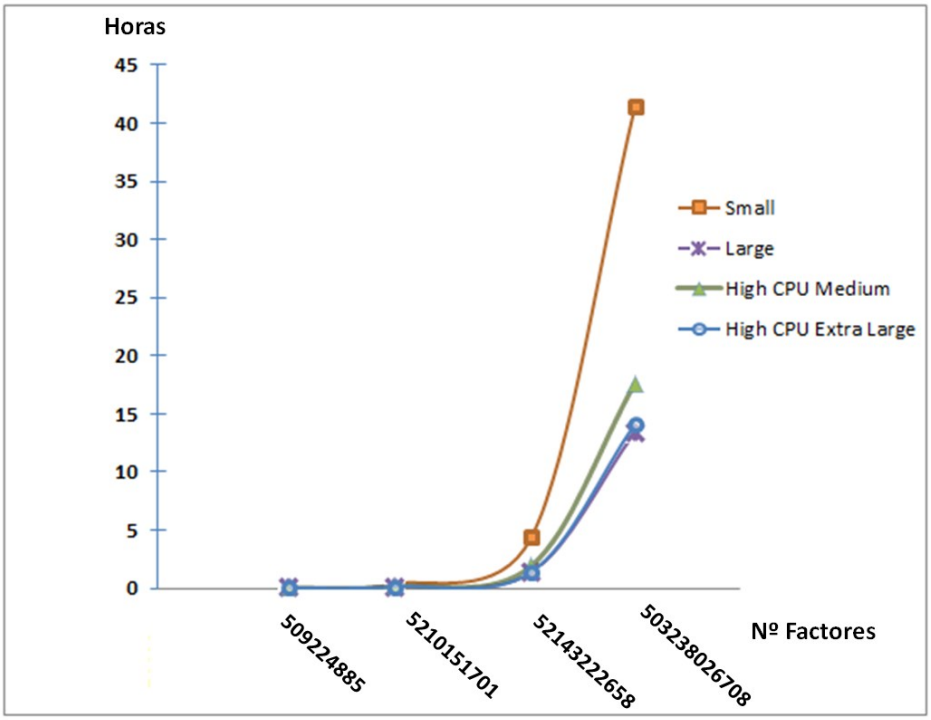


Figura 6.1: Tiempos de ejecución del algoritmo división por tentativa para diferentes tamaños de clave en las máquinas instanciadas de Amazon.

Las funciones vistas en la Figura 6.1 expresan la relación entre el número de factores que se deben probar para una clave determinada y el número de horas de ejecución.

Una vez definida la recta de regresión que describe el comportamiento del algoritmo en cada tipo de instancia para tareas individuales, se puede generalizar la ecuación del tiempo total *'T'* para el modelo de paralelización resultante de dividir el trabajo de factorizar cierta clave en un determinado número de subtareas, como ya explicamos en la sección 5.2.3. También, aparte del tiempo total de ejecución tendremos en cuenta otras métricas, como la denominada Coste/Rendimiento ($C/R$) óptima, también explicada en la misma sección.

Tras obtener el mejor $C/R$ para cada máquina, podemos evaluar cuál de ellas es mejor y averiguar el número de máquinas virtuales necesarias para el trabajo total en el mejor caso con la expresión vista en dicha sección.

El tamaño de cada tarea que ejecutará cada máquina remota viene determinado por la siguiente fórmula:

$$i = \frac{I}{k * N_{maqs} * N_{núcleos}} \qquad (Ec.\ 6.1)$$

Donde $I = \sqrt{N/2}$, que es la cantidad de números que comprueba el algoritmo.

En la ecuación Ec. 6.1, *'k'* es el número de tareas asignadas a cada núcleo de cada máquina. Tras considerar distintos valores para *'k'* llegamos a la conclusión que cuanto mayor sea su valor, mejor será el rendimiento de la máquina, ya que el tiempo de ejecución resultante '$T$' es menor. Esto deriva en un menor valor de *'i'* y por tanto más subtareas y transferencias de archivos de las mismas, por lo que se ha limitado el aumento del valor de *'k'* de tal manera que el producto $k * N_{núcleos}$ (número de tareas por máquina) sea 120 para evitar un exceso de tráfico de archivos resultado entre las máquinas, de tal forma que no se supere un margen de tres minutos permitido en el caso de que cada transacción implique un valor aproximado de un segundo de retardo. Se establece una franja de seguridad de un minuto, dejando dos minutos para transferencias entre máquinas.

Para valores de claves muy grandes, el número de instancias necesarias puede ser muy elevado, lo cual implica un problema: Amazon EC2 sólo permite instanciar un número máximo de 20 máquinas a la vez. Para la adquisición de más instancias ha de solicitarse a la empresa mediante un formulario justificando la necesidad de las máquinas.

En la Figura 6.2 se puede ver un ejemplo de búsqueda del tipo de instancia óptima para un valor de clave igual a 576956058084710807398269724 81 (29 dígitos). En ella observamos que a mayor uso de instancias mejor $C/R$. Pero a partir del momento en que se alcanza el valor óptimo de $C/R$, éste empieza a crecer, haciendo ineficiente el uso de más instancias.

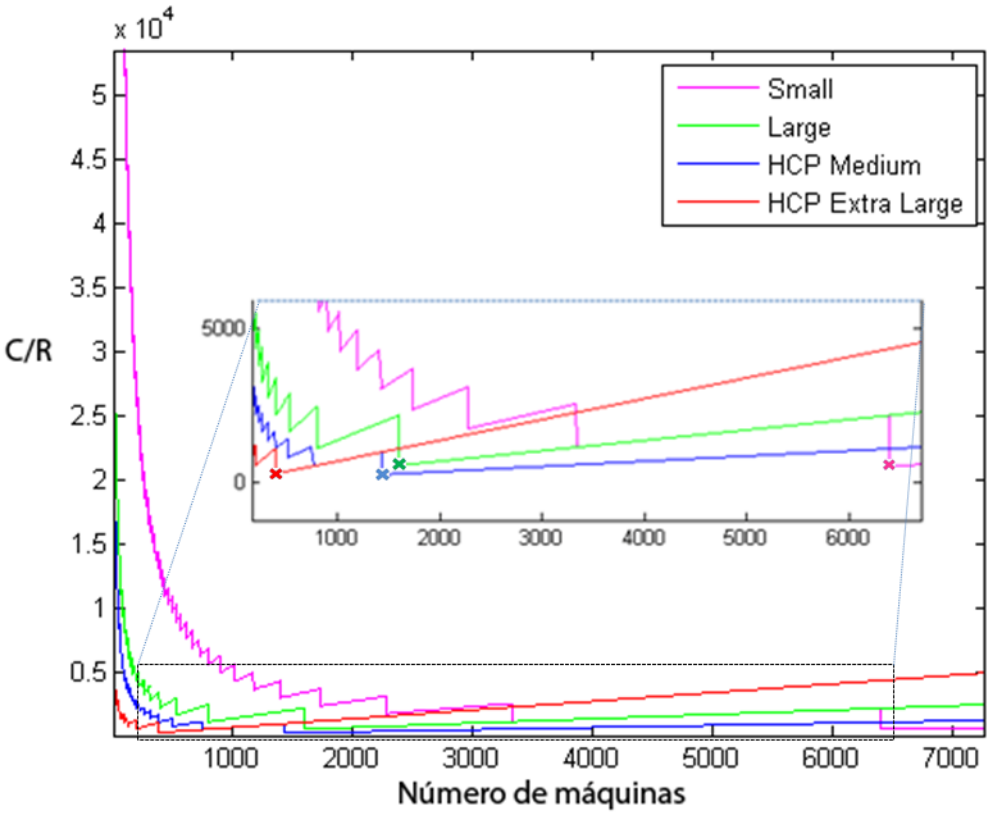


Figura 6.2: Comparación de $C/R$ para todas las instancias de Amazon EC2, para una clave igual a 576956058084710807398269724 81 y $k = 120$.

La gráfica interior de la Figura 6.2 ilustra los puntos de valor óptimo de $C/R$ para los cuatro tipos de instancias estudiadas.

Los valores mínimos de $C/R$ se alcanzan con diferente número de máquinas según el tipo de instancia elegido. La Figura 6.3 muestra los valores de $C/R$ y número de máquinas virtuales óptimos para el ejemplo considerado. Los valores del coste son muy aproximados al valor de $C/R$ obtenido ya que el tiempo está ajustado a, aproximadamente, una hora $(0.95\ horas = 57\ min)$.

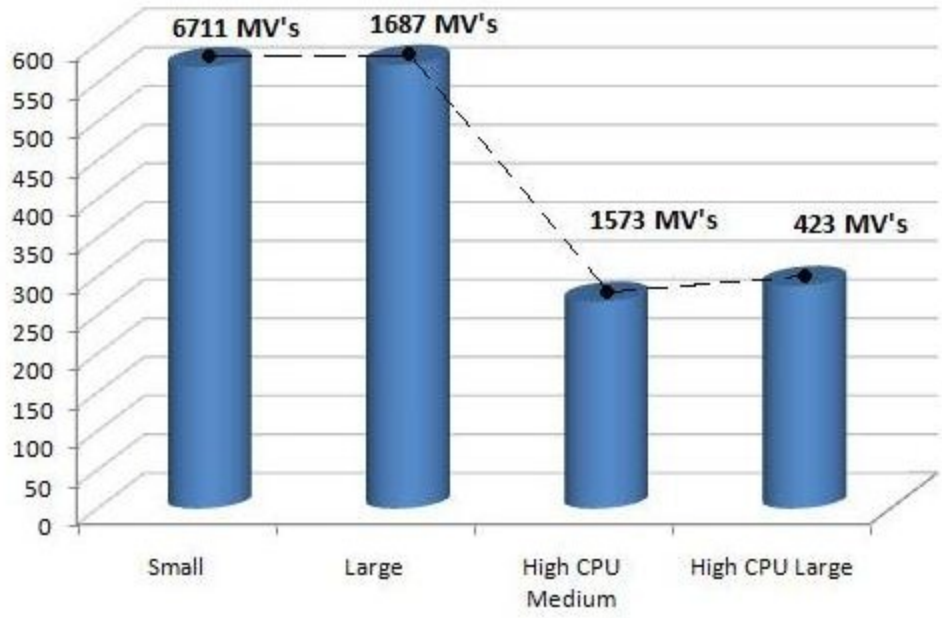


Figura 6.3: Comparación de $C/R$ óptimo fijando el tiempo de ejecución en 57 minutos para todas las instancias de Amazon EC2, para los mismos valores de clave y $'k'$ de la Figura X+1.

Analizando las instancias de tipo Small y Large se obtienen valores $C/R$ similares, pero el número de instancias de tipos Small cuadruplica al número de máquinas de tipo Large, debido a que el número de unidades de cálculo EC2 en la instancia Large es 4, mientras que la Small consta sólo de una unidad. Esta relación entre unidades de cálculo puede observarse también entre la familia de instancias de tipo "High CPU", con una proporción de unidades de cálculo de uno a cuatro, para la Medium respecto de la Extra Large.

Elegir un tipo de instancia se traduce a elección de pagar más por la infraestructura o disminuir el nivel de paralelismo. Viendo la Figura 6.3, la instancia de tipo Small es la solución más cara pero con el mayor número de núcleos trabajando simultáneamente $(6711)$. En la instancia tipo Large el número de núcleos es la mitad $(3374)$, pero con un coste similar. Las instancias de tipo "High CPU" realizan la misma tarea con un número de núcleos similar a la instancia Large, sin embargo el coste es mucho menor. En concreto el precio para High CPU Medium es $\$267{,}42$ y para High CPU Extra Large es $\$287{,}65$.

Observando los resultados obtenidos, determinamos que el tipo de instancia óptima es la High CPU Medium, debido al mayor aprovechamiento de la capacidad de cálculo de sus núcleos. Con respecto a las instancias Large y High CPU Extra Large, High CPU Medium consigue realizar la tarea de factorización con un coste más económico empleando un número de núcleos similar.

A continuación se detallan los informes de evaluación obtenidos con Forecaster:

Tipo Maquina: High CPU Medium
Numero Cores por Máquina: 2
Clave: 576956058084710807398269724 81
Intervalo Total I: 120099548092895
Intervalo por tarea i: 20016591348815
Numero Maquinas: 1
Tareas por Maquina: 6
Tareas por Core (k): 3
Tiempo de cada Tarea = 983.1343916469 ---> 983 Horas, 8 Minutos y 3 Segundos
Tiempo Total = 2949.40317495 ---> 2949 Horas, 24 Minutos y 11 Segundos
Coste/Hora: $ 0.17
Coste $ 501.51
Coste/Rendimiento (C/R) : 1479454.5

Tipo Maquina: High CPU Medium
Numero Cores por Máquina: 2
Clave: 576956058084710807398269724 81
Intervalo Total I: 120099548092895
Intervalo por tarea i: 636255287
Numero Maquinas: 1573
Tareas por Maquina: 120
Tareas por Core (k): 60
Tiempo de cada Tarea = 0.0158236419 ---> 0 Horas, 0 Minutos y 56 Segundos
Tiempo Total = 0.94941852 ---> 0 Horas, 56 Minutos y 57 Segundos
Coste/Hora: $ 0.17
Coste $ 267.42
Coste/Rendimiento (C/R) : 267.42

De los informes obtenemos que la mejora de $C/R$ es significativa: mientras que con una máquina obtenemos un $C/R$ de 147954.5, trabajando con la métrica $C/R$ óptima, esto es 1573 máquinas, obtenemos un $C/R$ de 267.42.

# 6.2 Caso de uso: Forecaster Vs. Engine

## 6.2.1 Introducción

Tras la implementación del módulo Forecaster, y partiendo de los cálculos teóricos descritos en la sección anterior, se requiere realizar una estimación en términos de tiempo, coste y C/R de una tarea de factorización y realizar un experimento real con instancias de Amazon EC2 con el fin de verificar la validez de esta herramienta. Este ejemplo de ejecución también sirve un pequeño tutorial a modo de manual de uso (para ver todos los detalles de su funcionamiento, ver la sección 5.3.3).

## 6.2.2 Configuración del experimento

Se ha escogido una clave RSA a factorizar de 23 dígitos:

13395808278084116677277

El tamaño de la clave es premeditado, de modo que la tarea total no excede el tiempo de ejecución de una hora, como veremos más adelante. Esto es así para que el coste de la prueba sea pequeño, pues únicamente se quiere demostrar que la aplicación es válida.

Las máquinas a emplear son 5 instancias Small (familia estándar) de AWS Amazon EC2. Sus características se pueden ver en la sección 2.2.2.

Cada máquina va a ejecutar 3 tareas. Es un número muy reducido pero suficiente para este caso, dado que la clave no es excesivamente grande. El principal motivo de esta decisión es para mostrar que el tamaño de intervalo parcial *'i'* puede ser un parámetro introducido manualmente, en lugar de activar la opción para que el programa lo escoja de forma automática. Según la expresión (Ec.5.3) de la sección 6.1, la ecuación del intervalo parcial *'i'* para este caso es:

$$i = \frac{I}{k * N_{maqs} * N_{núcleos}} = \frac{57870131065}{3 * 5 * 1} = 3858008737{,}667$$

Donde $I = \frac{\sqrt{N}}{2} = \frac{\sqrt{13395808278084116677277}}{2} = 57870131065{,}3520$, que es la cantidad de números que comprueba el algoritmo. Por tanto, el intervalo parcial *'i'* será la parte entera del resultado:

$$i = 3858008737$$

### 6.2.3 Forecaster: Estimación

Ahora que tenemos todos los parámetros de entrada, se inicia el módulo Forescaster haciendo doble clic en el ejecutable con extensión '.jar'. En el menú inicial, se selecciona el modo 'Manual' para poder establecer todos los parámetros de entrada.


Figura 6.4: Menú de inicio.

Tras pulsar el botón 'Aceptar' aparece la ventana principal y se introducen valores para todos los campos disponibles en el panel izquierdo según los datos anteriormente expuestos de la siguiente manera:

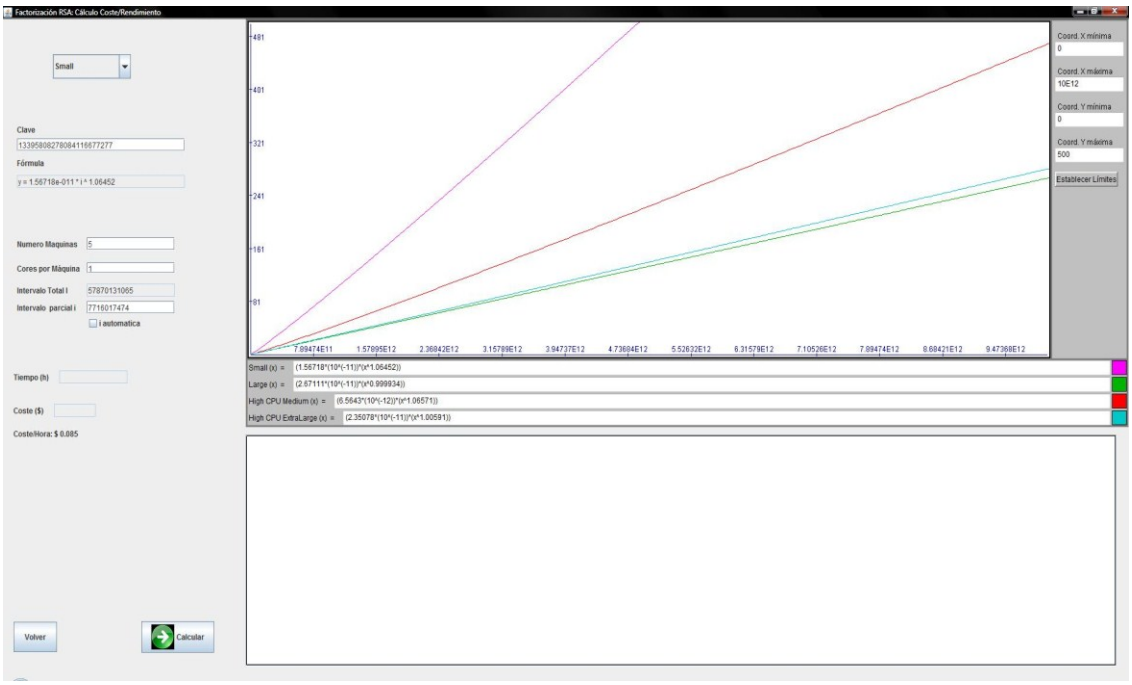


Figura 6.5: Ventana principal. Antes de presionar el botón 'Calcular', se rellenan todos los campos con los datos de entrada.

A continuación se procede al cálculo de la estimación para la configuración introducida, y seguidamente se muestran los resultados:

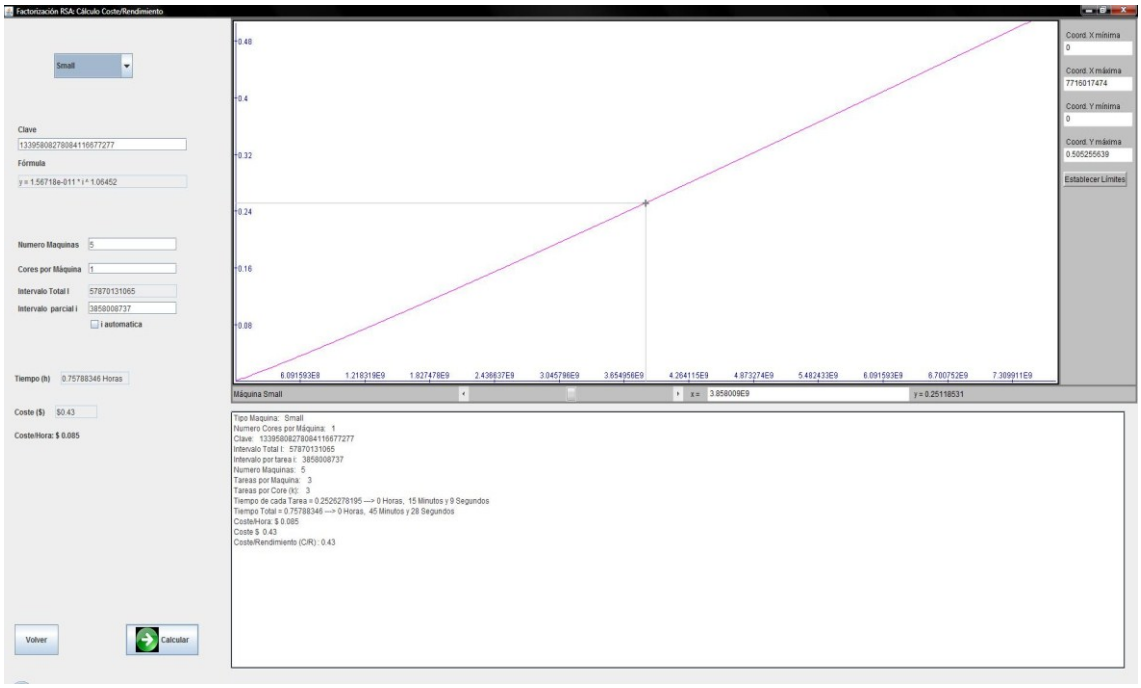


Figura 6.6: Ventana principal. Tras pulsar el botón 'Calcular', la aplicación muestra los resultados de estimación y la gráfica para la función de tiempo de ejecución para el intervalo parcial *'i'*.

La salida del panel inferior de resultados muestra los siguientes datos:

- Tipo Maquina: Small
- Número Cores por Máquina: 1
- Clave: 13395808278084116677277
- Intervalo Total I: 57870131065
- Intervalo por tarea i: 3858008737
- Número Maquinas: 5
- Tareas por Maquina: 3
- Tareas por Core (k): 3
- Tiempo de cada Tarea = 0.2526278195 ---> 0 Horas, 15 Minutos y 9 Segundos
- Tiempo Total = 0.75788346 ---> 0 Horas, 45 Minutos y 28 Segundos
- Coste/Hora: $ 0.085
- Coste $ 0.43
- Coste/Rendimiento (C/R): 0.43

El dato más importante que se debe tener en cuenta para el experimento práctico posterior es que el tiempo de ejecución total de factorizar la clave '13395808278084116677277' es de 45 minutos y 28 segundos, según la estimación del módulo Forecaster.

### 6.2.4 Engine: Caso Práctico

Para proceder a ejecutar el algoritmo de división por tentativa de forma paralela entre las 5 máquinas de Amazon, obtenemos su IP pública en la cuenta propia de AWS Amazon EC2. Posteriormente en la máquina cliente, encargada de distribuir las tareas, concretamente en el directorio donde se encuentra ubicado el módulo Engine, se configuran todos los archivos de la carpeta "datos" de la siguiente manera:

**- algoritmo.txt**: Para este algoritmo, en este archivo se escribe *trialdiv*.

**- clave.txt**: Se introduce en una sola línea la clave *13395808278084116677277*.

**- maquinas.txt:** En cada línea se incluye la información relativa a una máquina que actuará como servidor. Para este caso, en este fichero hay 5 líneas escritas con una expresión de la siguiente forma:

<IPMáquina> LIBRE <usuario> <contraseña>

**- rangos.txt:** Se ejecuta la instrucción "perl creaRangos.pl" para definir el rango de números que abarca cada tarea.

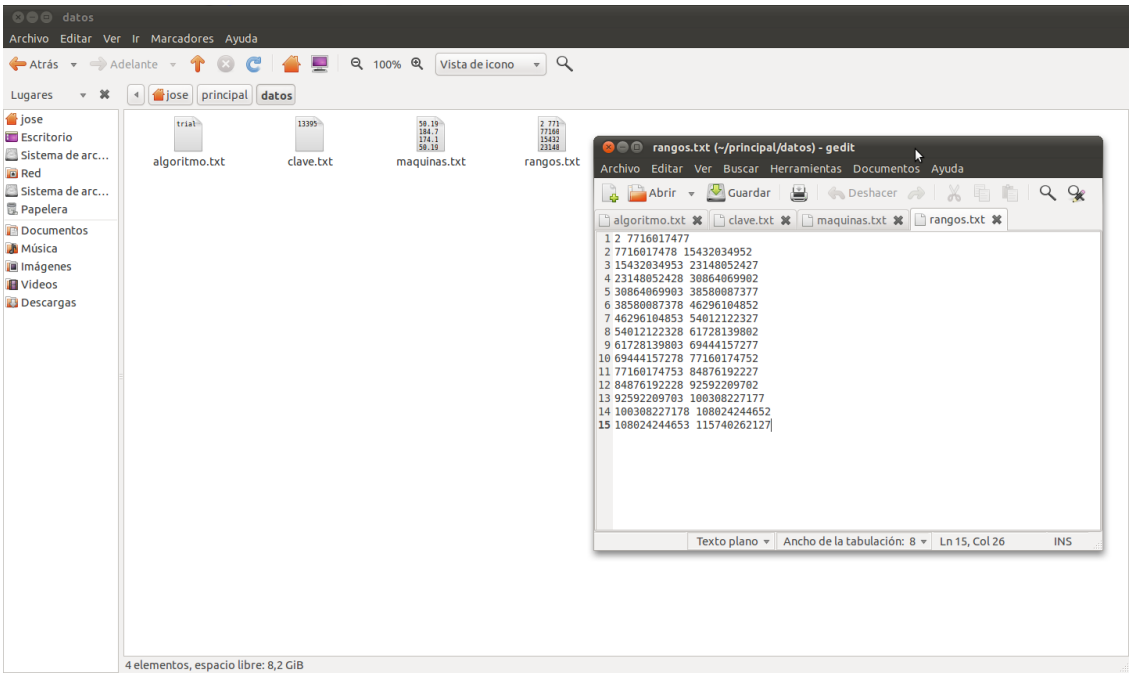


Figura 6.7: Establecimiento de los parámetros de entrada en la carpeta 'datos'.

Ahora sólo resta ejecutar los scripts Perl 'conexion.pl' y seguidamente 'tareas.pl' para comenzar con el ataque a la clave RSA. La máquina cliente comienza a crear archivos de datos de subtareas y los manda a las 5 instancias de Amazon. Al cabo de un tiempo éstas devuelven los ficheros de texto con el resultado de cada subtarea. Este proceso se repetirá 3 veces por cada máquina, que corresponde al número de subtareas ejecutadas por máquina. Cuando se termina, la carpeta 'resultados' tiene esta apariencia:

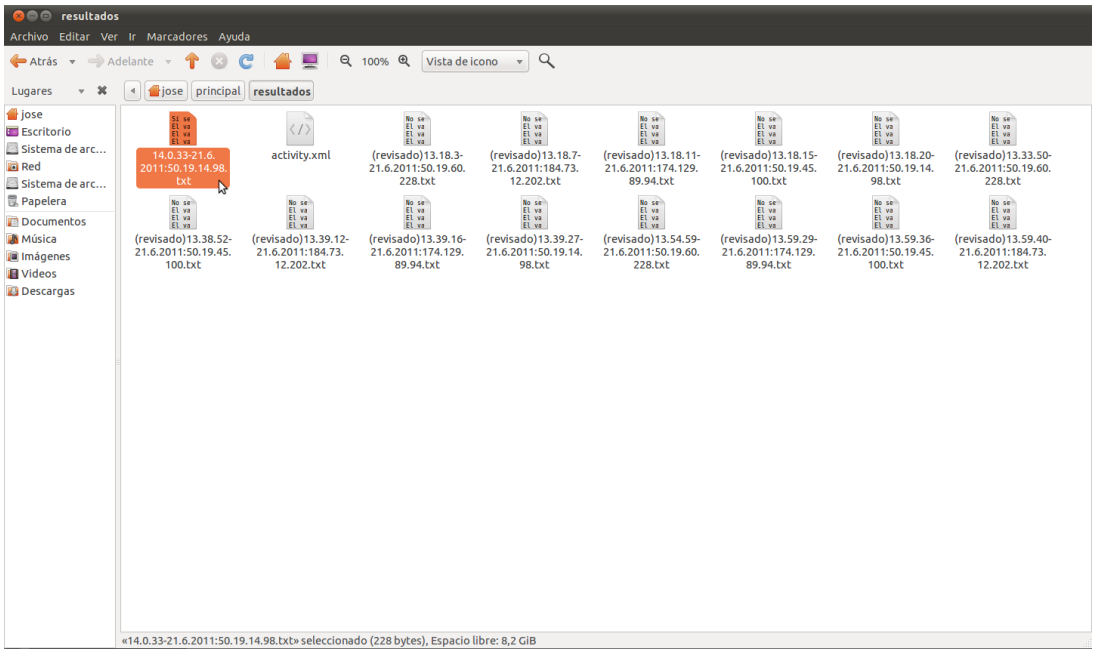


Figura 6.8: Estado del directorio resultados al finalizar la ejecución.

Al final de la ejecución del script 'tareas.pl', la máquina cliente ha recibido los resultados de todas las subtareas ejecutadas hasta haber encontrado un factor de la clave, y ha generado un archivo log de actividad 'activity.xml'.

Cuando se abre el fichero resultado correspondiente a la subtarea que ha dado con uno de los factores de la clave, se puede leer lo siguiente:

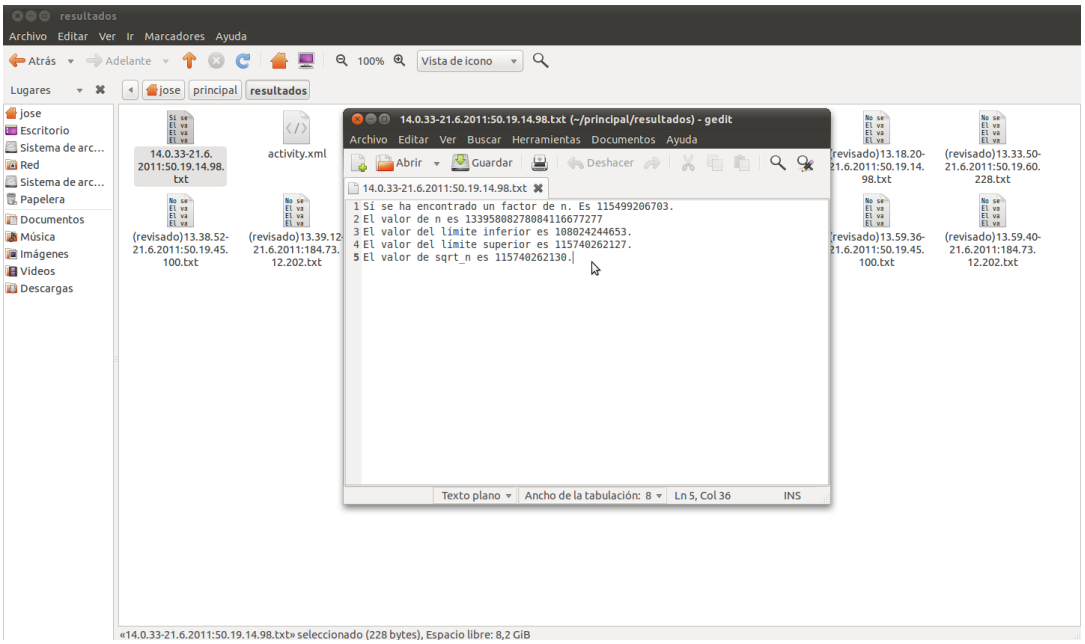


Figura 6.9: Contenido del fichero resultado con la solución al problema de factorización.

En este archivo podemos observar que el factor encontrado para la clave de 23 dígitos es '**115499206703'**. En efecto:

133958082780841166772777 = 115981820659 * 115499206703

Una vez que se ha confirmado que la clave se ha descifrado correctamente, se procede a revisar el tiempo de ejecución total empleado. Para ello se procede a averiguar:

- **La fecha de creación de la primera subtarea** enviada a la primera máquina de la lista del archivo 'maquinas.txt'. Esta máquina contiene el archivo con los datos de la primera subtarea llamado 'infoExec0.txt' en el directorio de trabajo. Se anotará su fecha de creación y la consideraremos la fecha del inicio de la ejecución, pues se desprecia el tiempo transcurrido hasta el momento en que la máquina recibe el archivo procedente del cliente (su tamaño es de sólo 4KB) desde que se inicia el proceso. Se anota la IP pública de dicha máquina y se procede a la conexión SSH manualmente. Dentro de la ruta donde se ubica el archivo, se procede a obtener su fecha de creación. A continuación se muestra la salida de los comandos "stat" y "ls -lc":

```
  user@ip-10-36-9-61:~/principal$ stat infoExec0.txt
    File: `infoExec0.txt'
    Size: 55                Blocks: 8          IO Block: 4096
regular file
  Device: ca01h/51713d      Inode: 131154      Links: 1
  Access: (0644/-rw-r--r--)  Uid: ( 1001/      user)     Gid: (
1001/    user)
  Access: 2011-06-21 13:18:03.699944010 +0000
  Modify: 2011-06-21 13:18:00.635944320 +0000
  Change: 2011-06-21 13:18:00.635944320 +0000

-----------------------------------------------------------------

  user@ip-10-36-9-61:~/principal$ ls -lc infoExec0.txt
  -rw-r--r-- 1 user user 55 2011-06-21 13:18 infoExec0.txt
```

- **El nombre del último fichero resultado** recibido por la máquina cliente con el factor de la clave. En la carpeta 'resultados' del directorio de trabajo de la máquina cliente, el nombre de los ficheros resultados tienen la forma:

<hora>-<fecha>:<IP_servidor>.txt

En este caso, el nombre del archivo es:

14.0.33-21.6.2011:50.19.14.98.txt

Con las fechas de inicio y final de ejecución, se puede deducir el tiempo total aproximado empleado por el módulo Engine en factorizar la clave RSA, que es igual a:

Tiempo Total = 42 minutos y 33 segundos.

### 6.2.5 Resultados obtenidos

La ejecución del Engine dejó los siguientes resultados:

- Número de archivos de tareas generadas/archivos resultado recibidos: 15.
- Tamaño del archivo 'activity.xml': 36 KB.
- Tiempo total del proceso de factorización: 42 minutos y 32 segundos.

Comparando el tiempo estimado por el módulo Forecaster (**45 minutos y 28 segundos**) y el tiempo real empleado por el módulo Engine (**42 minutos y 33 segundos**), calculamos el error absoluto y relativo de la estimación:

Estimación Forecaster en horas: $T_{estimado} = 0{,}758$ h.

Tiempo total de ejecución en horas: $T_{real} = 0{,}701$ h.

Error absoluto: $\mathrm{E_{abs}} = |T_{estimado} - T_{real}| = |0{,}758 - 0{,}701| = 0{,}057$

Error relativo: $\mathrm{E_{rel}} = \frac{\mathrm{E_{abs}}}{T_{real}} = \frac{0{,}057}{0{,}758} = 0{,}075 = 7{,}5\ \%$

### 6.2.6 Valoración

Este ejemplo sirve para confirmar que la herramienta Forecaster ofrece una estimación aproximada del tiempo que se necesitaría para factorizar una clave dada. Analizando el resultado, observamos que las estimaciones del Forecaster se realizan por exceso y no por defecto, brindando una cota máxima de tiempo y dinero para que el usuario no se encuentre con sorpresas a la hora de revisar la factura de Amazon.

Los 4 puntos empleados en la obtención de estas funciones (ver sección 6.1) no permiten obtener aproximaciones más exactas, pero eso puede perfeccionarse con el cálculo de nuevos puntos que faciliten la obtención de funciones que resulten en un error relativo menor.

En resumen, los resultados son satisfactorios porque demuestran el potencial de RSA@Cloud y se establecen las bases de un sistema que cuente con una herramienta de estimaciones fiable y módulo descifrador de claves de gran proyección.

# 6.3 Aplicación Engine - Ataque a distintas claves

## 6.3.1 Introducción

En esta sección revisamos a fondo un experimento real con la herramienta Engine para la factorización de dos módulos RSA de 39 y 49 dígitos con el algoritmo de Criba Cuadrática.

## 6.3.2 Configuración del experimento

La clave a factorizar utilizada para este experimento es

106707255660663370356488196961408690567

Disponemos de dos máquinas que actuarán como servidores, cuyas características son:

- PC de mesa: Procesador AMD Phenom 9550 Quad Core @ 2.2 GHz. Sistema operativo 32 bits. Cuatro núcleos físicos.
- Ordenador portátil: Procesador Intel(R) Core(TM) i3CPU M330@ 2.13 GHz, RAM: 4 GB (3.86 GB utilizable). Sistema Operativo de 64 bits. Dos núcleos físicos. Hemos aprovechado las características de Hiperthreading (con dos núcleos físicos se obtiene el rendimiento de cuatro núcleos virtuales) que ofrece el procesador i3 para obtener tres núcleos de este ordenador. No utilizamos los cuatro posibles ya que saturamos la memoria principal y obtenemos un peor rendimiento.

## 6.3.3 Resultados obtenidos

Los parámetros para la configuración de la Criba Cuadrática obtenidos son los siguientes:

- Número de elementos de la Base de Factores: 1.092.
- Tamaño del Intervalo de Criba: 2.610.491.638.

La ejecución del Engine dejó los siguientes resultados:

- Número de archivos de tareas generadas/archivos resultado recibidos: 73.
- Tiempo total del proceso de criba: 12 minutos 1 segundo.
- Tamaño del archivo 'Resultados.txt': 170 KB.

- Tiempo del módulo de álgebra lineal: 0 segundos.
- Tiempo total de la Criba Cuadrática: 12 minutos 1 segundo.
- Tamaño del archivo 'activity.xml': 154 KB.

El resultado final es:

La primera solución devolvió la factorización no trivial
Los factores son:
106707255660663370356488196961408690567 = 10168938831019335571 * 10493450440980479677

### 6.3.4 Configuración del experimento

La clave a factorizar utilizada para este experimento es

101954124306182613785148212144352675448171552537 3

Disponemos de siete máquinas que actuarán como servidores, de las cuales:

- 5 máquinas virtuales Small (familia estándar) pertenecientes al Cloud público de Amazon. Sus características se pueden ver en la sección 2.2.2.
- 2 máquinas personales, cuyas características pueden verse en el ejemplo anterior. Para este experimento hemos utilizado los cuatro núcleos físicos del PC de mesa.

### 6.3.5 Resultados obtenidos

Los parámetros para la configuración de la Criba Cuadrática obtenidos son los siguientes:

- Número de elementos de la Base de Factores: 3.176.
- Tamaño del Intervalo de Criba: 64.083.703.940.

La ejecución del Engine dejó los siguientes resultados:

- Número de archivos de tareas generadas/archivos resultado recibidos: 2.659.
- Tiempo total del proceso de criba: 2 horas 25 minutos 48 segundos.
- Tamaño del archivo 'Resultados.txt': 531 KB.
- Tiempo del módulo de álgebra lineal: 3 segundos.
- Tiempo total de la Criba Cuadrática: 2 horas 25 minutos y 51 segundos.
- Tamaño del archivo 'activity.xml': 5.83 MB.

El resultado final es:

La primera solución devolvió la factorización no trivial.
Los factores son:

10195412430618261378514821214435267544817155253 73 =
100317822664330286613252 7 * 10163111758049962356076 99

## 6.3.6 Capturas de los experimento

maquinas.txt

```
1 85.53.217.29 LIBRE tonio user1234
2 85.53.217.29 LIBRE tonio user1234
3 85.53.217.29 LIBRE tonio user1234
4 217.125.176.234 LIBRE jose user1234
5 217.125.176.234 LIBRE jose user1234
6 217.125.176.234 LIBRE jose user1234
7 217.125.176.234 LIBRE jose user1234
8 50.17.63.32 LIBRE user user1234
9 174.129.60.94 LIBRE user user1234
10 50.19.146.249 LIBRE user user1234
11 184.72.131.124 LIBRE user user1234
12 50.19.128.7 LIBRE user user1234
```

Figura 6.10: Contenido del archivo 'maquinas.txt' para el experimento de 49 dígitos (12 núcleos).

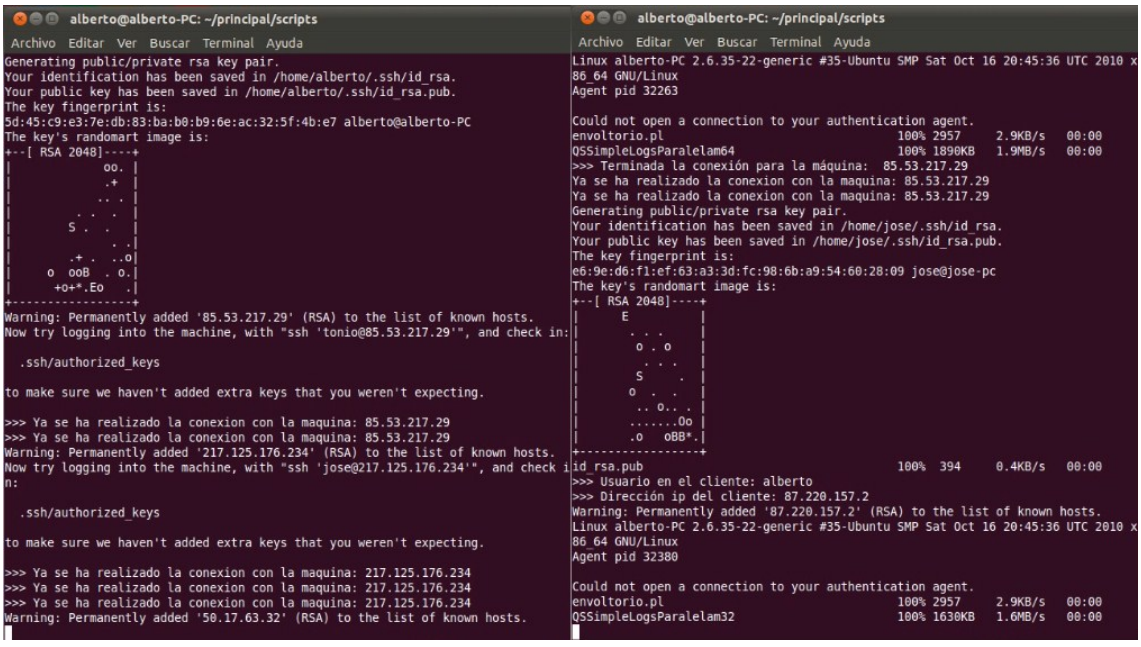


Figura 6.11: Vista desde el cliente – Etapa de conexión.

En la Figura 6.11 podemos ver la consola del cliente durante la ejecución de la conexión con los servidores. En la parte de la izquierda se observa la generación de claves y que cuando se trabaja con más de un núcleo de una misma máquina sólo se establece la conexión una vez. A la derecha vemos el envío del envoltorio y del algoritmo de factorización dependiendo de la arquitectura de la máquina: 32 o 64 bits.

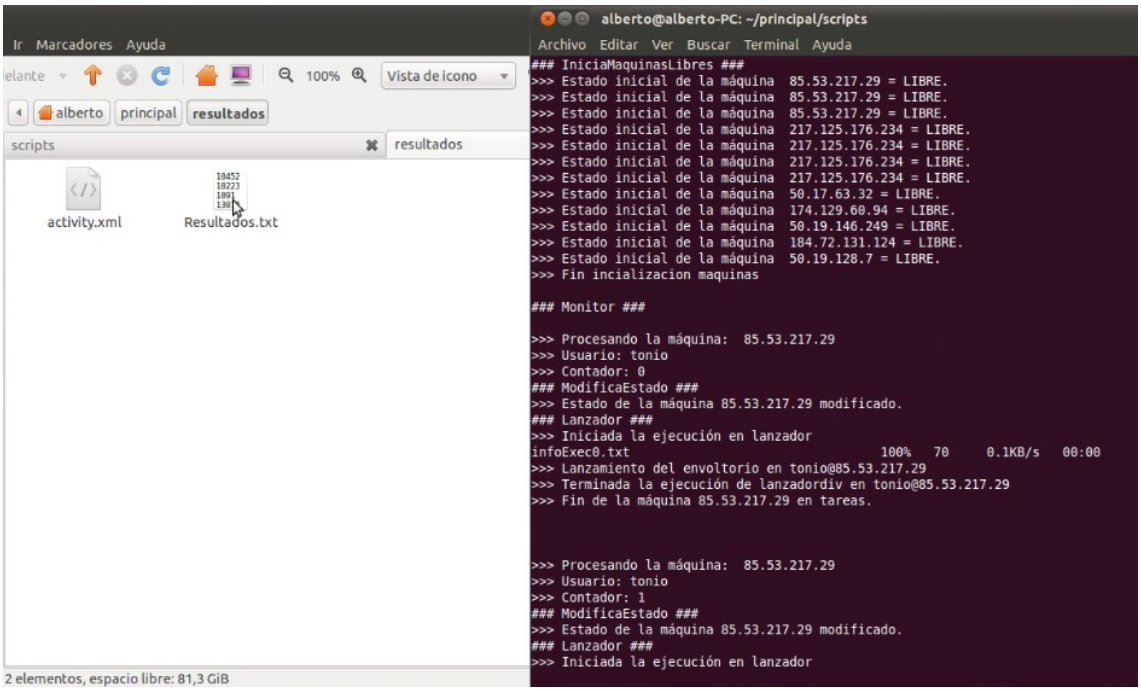


Figura 6.12: Vista desde el cliente – Etapa de ejecución (inicio).

En la figura 6.12 podemos ver el inicio del módulo de ejecución. En la carpeta de resultados, se ha generado el archivo 'activity.xml' y 'Resultados.txt', con la información oportuna para el Codeswarm y el algoritmo de Criba Cuadrática, respectivamente. En la consola se observa cómo se inicializa el estado de las máquinas a LIBRE y la distribución de tareas entre los servidores.

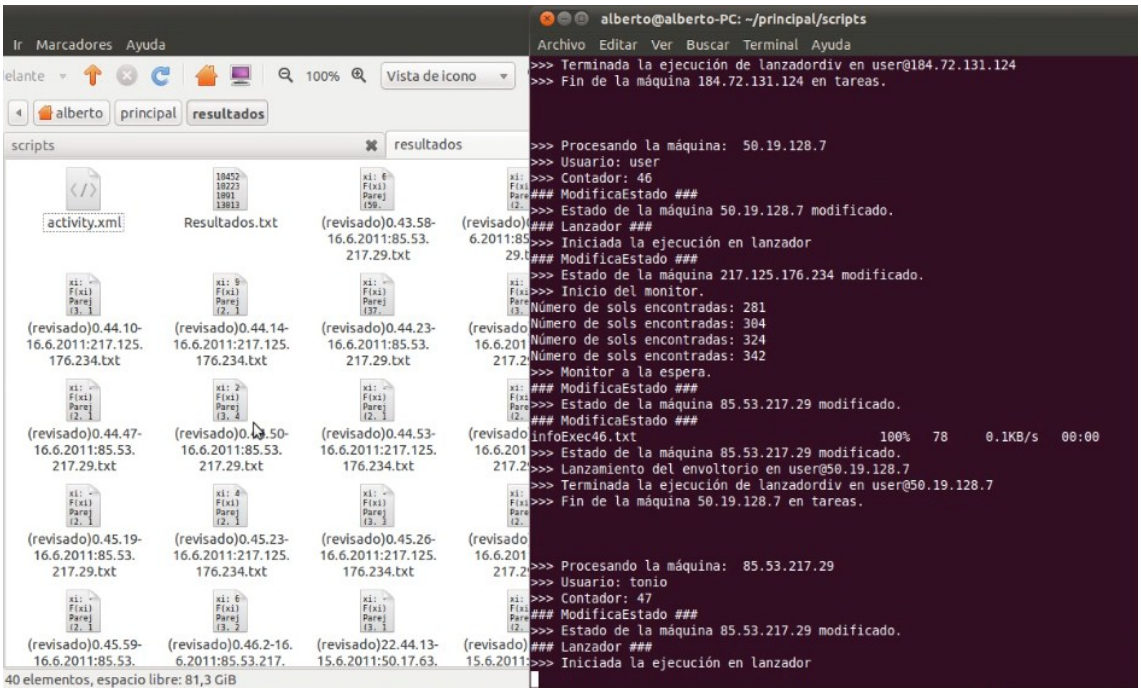


Figura 6.13: Vista desde el cliente – Etapa de ejecución (desarrollo).

En la Figura 6.13 vemos el desarrollo de la ejecución desde el cliente. El cliente va recibiendo los archivos de resultados generados por cada servidor y los revisa, añadiéndolos a 'Resultados.txt'. En la consola observamos que el monitor entra periódicamente a revisar la carpeta, mostrando el número de soluciones que ha encontrado hasta este momento.

A continuación mostramos capturas del video generado por Codeswarm para los experimentos realizados:

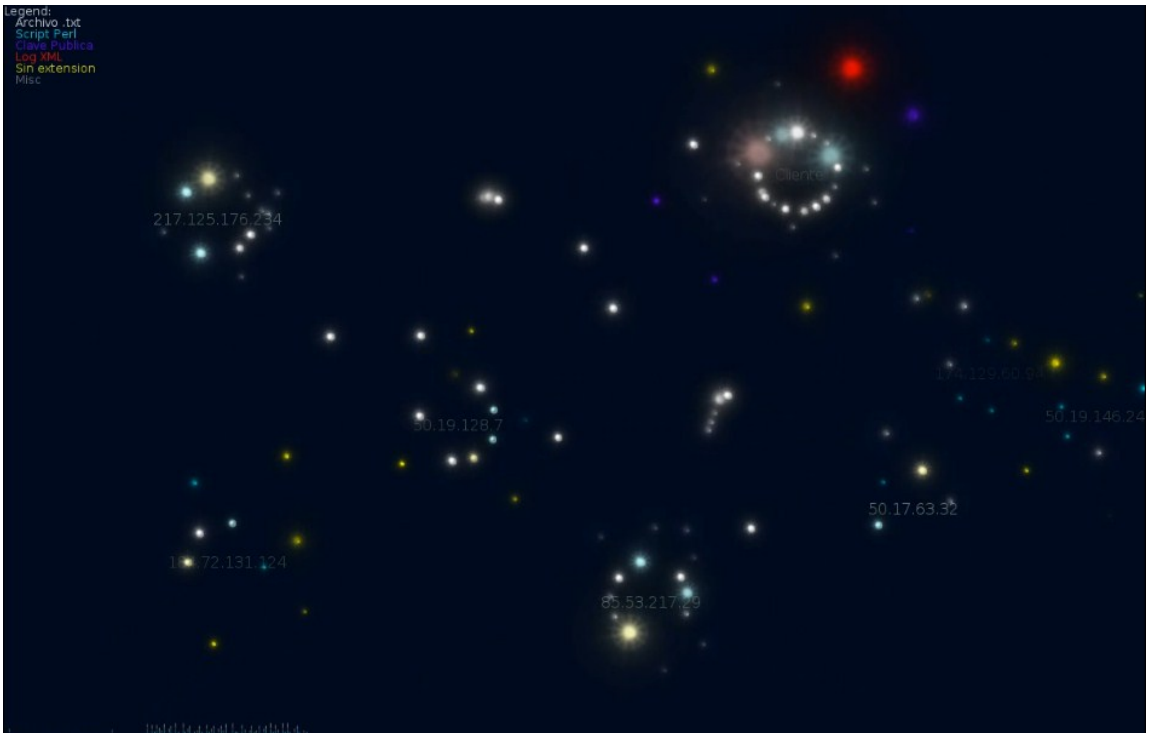


Figura 6.14: Codeswarm – Fase de criba para la clave de 39 dígitos con 11 núcleos activos.

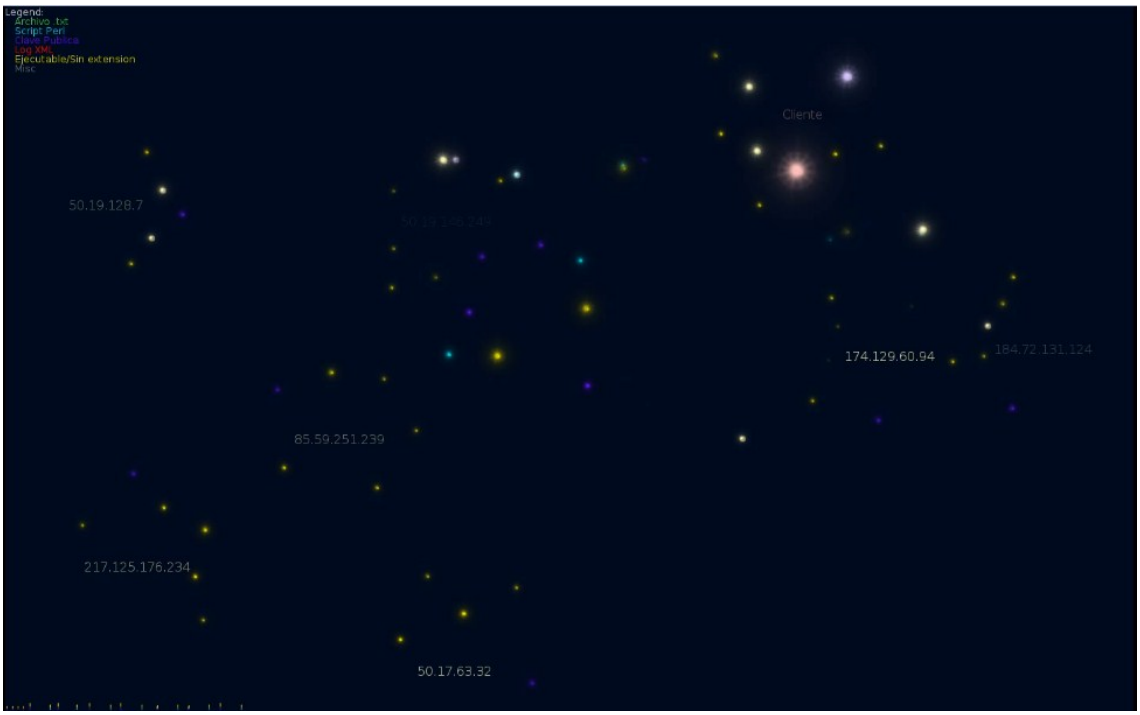


Figura 6.15: Fase de conexión para la clave de 49 dígitos con 12 núcleos activos.

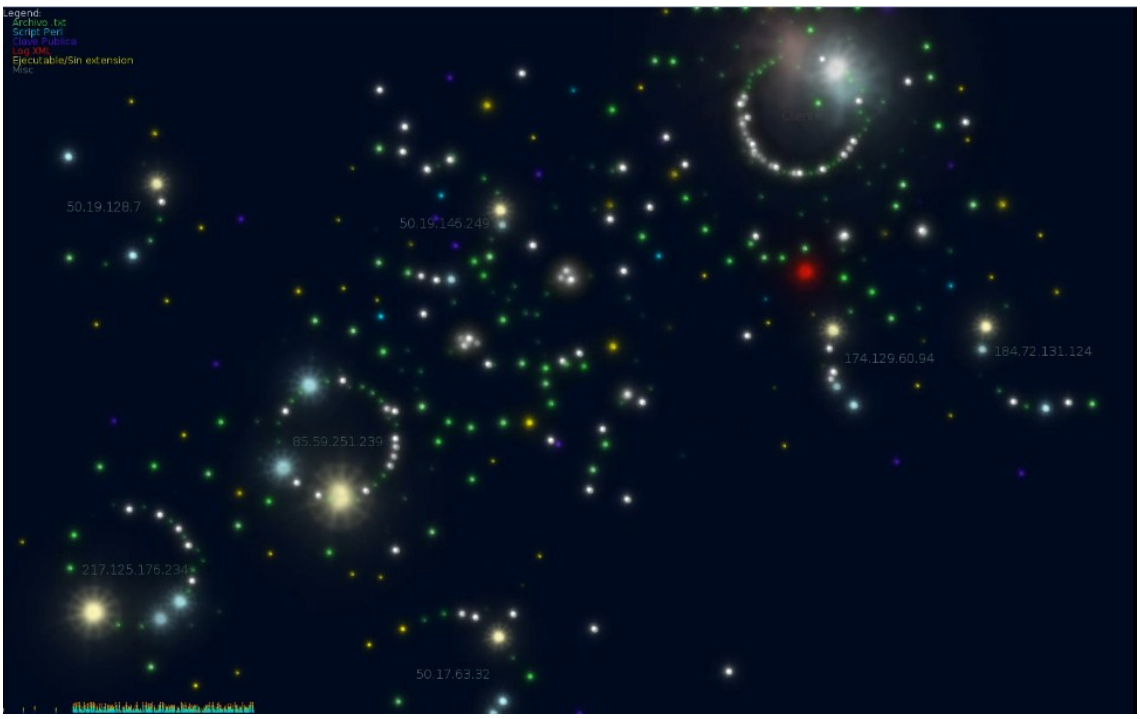


Figura 6.16: Fase de ejecución para la clave de 49 dígitos con 12 núcleos activos.

# 7. Trabajo futuro

## 7.1 Introducción

Los proyectos de investigación científica son siempre mejorables. A pesar de que la meta del progreso y del conocimiento científico es la perfección, ésta nunca es alcanzable. Pero el simple hecho de fijar ese objetivo produce siempre mejores resultados, imperfectos, pero más cercanos a la perfección. Este espíritu es el que ha conducido a la sociedad al nivel tecnológico y cultural de hoy en día. Enfocándonos ahora en este trabajo, con sus virtudes y sus limitaciones, este sistema es el producto de ardua tarea de investigación en el que no se han impuesto límites en el transcurso de su construcción.

A lo largo de todas las etapas de diseño, planificación, desarrollo y análisis del proyecto, se han realizado cambios con vista a mejorar nuestro sistema, ya sea en términos de eficiencia, usabilidad, tiempo, modularidad, etc., pero muchas otras ideas se han quedado en el camino, bien por falta de recursos, de tiempo, o incluso de conocimientos. Es por estas razones que sería muy provechoso y estimulante enumerar esas ideas de ampliación que el grupo ha planteado en cada etapa del proyecto o aquellas que, a posteriori, una vez finalizado el desarrollo y obtenido las conclusiones, han surgido como si de un *"brainstorm"* o tormenta de ideas se tratara. Unas son más factibles para ser investigadas por este mismo grupo a partir del sistema descrito en este documento y contando con escasos recursos, otras supondrían contar con un colectivo en el que estuvieran involucrados una cantidad mayor de miembros de proyecto, con grandes conocimientos en muchos ámbitos de la informática y las matemáticas: desde la infraestructura de red hasta la criptografía.

Con vistas a continuar la investigación de algoritmos de factorización sobre infraestructuras Cloud, el sistema desarrollado dispone de un alto potencial en cuanto a posibilidades de ampliación, gracias a que el módulo principal Engine está diseñado de forma modular, lo cual permite la inclusión de, entre otras funcionalidades que veremos a continuación, nuevos algoritmos de factorización.

Sin más preámbulos, a continuación se enumeran y describen las posibles opciones de trabajo futuro que plantea este proyecto, RSA@Cloud, tanto a este grupo como a cualquier otro interesado en la computación en la Nube, criptosistemas, paralelismo y sistemas distribuidos. Éstas son nuestras propuestas:

# 7.2 Implementación del algoritmo GNFS

Existen distintos algoritmos matemáticos destinados a factorizar números de aún mayor magnitud que la división por tentativa y la criba cuadrática. Uno de los algoritmos más interesantes para esta tarea sería el algoritmo denominado GNFS ("General Number Field Sieve" o Criba General del Cuerpo de Números). Este algoritmo es actualmente el algoritmo clásico (no cuántico) más eficiente a la hora de descomponer números enteros de más de 110 dígitos (ver sección 4.7). Gracias a la modularidad de la plataforma, se podrían desarrollar la implementación de éste u otros algoritmos de factorización de propósito especial o general capaces de aprovechar los beneficios de la Nube para la paralelización de sus cálculos.

El trabajo de incorporar una implementación de GNFS al sistema supondría crear el código desde cero, o bien aprovechar la modularidad de nuestro sistema y escoger algún proyecto informático ya finalizado y testado, para luego realizar las modificaciones necesarias para amoldarse a nuestro sistema, convirtiéndolo en paralelizable. A corto o medio plazo, ésta sería la solución más factible.

Actualmente ya existe alguna implementación de este algoritmo en código abierto desde su publicación en 1988. Un ejemplo de ello es el proyecto **pGNFS**[48], implementado por P. Leslie en su tesis *"Integer Factorization"* en 2005. El código fuente es totalmente descargable desde su sitio web. Este proyecto está escrito en C++, por lo que los recursos que emplea no son muy diferentes de los utilizados para crear el código del algoritmo de la criba cuadrática.

En concreto, este programa emplea dos librerías matemáticas para C++, una de las cuales ya se ha empleado en nuestro proyecto. Son las librerías NTL y GiNac[49]. Ha sido testado en diferentes plataformas Linux, pero dado que el lenguaje C++ es multiplataforma, pGNFS también debería serlo para todos los sistemas operativos en los que ambas librerías, NTL y GiNaC, estén disponibles.

GiNaC está diseñado para permitir la creación de sistemas integrados que incorporen manipulaciones simbólicas, o bien sistemas que pertenezcan a áreas más consolidadas de la informática (como la computación de alto nivel, aplicaciones numéricas, interfaces gráficas, etc.) bajo un mismo techo. Se distribuye bajo los términos y condiciones de la licencia "GNU General Public License" (GPL), del mismo modo que NTL, Codeswarm, JCM o cualquiera de los componentes empleados en este proyecto. Sin embargo, no se limita a aplicaciones de alta energía física. Su diseño es revolucionario en el sentido de que a diferencia de otros CAS ("Computer Algebra System") no intenta proporcionar amplias capacidades de álgebra y un lenguaje de programación simple, sino que acepta un lenguaje dado (C++) y lo extiende con un conjunto de capacidades algebraicas. Volviendo a pGNFS, su funcionamiento es relativamente sencillo una vez se estudie en profundidad el algoritmo GNFS (para más información al respecto, ir al punto 4.7 de este mismo documento, donde se da una visión general de la Criba General del Cuerpo de Números).

[48] http://pgnfs.org/
[49] http://www.ginac.de

Para compilarlo, basta una simple instrucción "make all". Antes de ejecutarlo, hace falta crear dos ficheros: un archivo con los parámetros de entrada al programa y un archivo que contenga al menos los números primos hasta el límite especificado para las bases en el archivo de entrada. El código está en su versión v0.3 y aún es mejorable, pero su funcionamiento y modo de uso son parecidos a nuestro sistema, lo que lo convierten en una opción interesante.

**kmGNFS**[50]es otro ejemplo de un proyecto de implementación del algoritmo de la Criba General del Cuerpo de Números(GNFS) escrito en C++. Su desarrollo comenzó en 2008 como parte de la tesis de máster de los ingenieros informáticos Christos Bakogiannis y Karapanos Nikolaos.

Este código también es absolutamente software libre escrito en C++. Hace uso únicamente de la librería NTL, lo que supone una gran ventaja con respecto al ejemplo anterior. Aparte de esto, kmGNFS utiliza principalmente C++ estándar, y debería compilarse perfectamente en cualquier sistema UNIX. Se ha probado bajo Linux, FreeBSD y Mac OS X. La versión distribuida de kmGNFS utiliza MPI, y ha sido probado en un entorno de MPI ("Message Passing Interface", un protocolo de comunicación entre máquinas a través de mensajes).

Por último, **GGNFS**[51]es otra implementación del algoritmo de la Criba General del Cuerpo de Números (GNFS) para factorizar enteros, desarrollada por el Doctor en Matemáticas Chris Monico. Su desarrollo se ha estancado en los últimos años, pero se ha probado para números de más de 180 dígitos con SNFS y hasta 140 números en el método general.

Algunos números más grandes se han factorizado también, pero hay problemas en el software que hacen más difícil la finalización correcta del algoritmo en el mejor de los casos. En su mayor parte, el rango superior en la que GGNFS es bastante confiable es, probablemente, hasta alrededor de 150 a 160 dígitos para SNFS, y 130-135 con el método general GNFS. Tiene ciertos errores, pero casi en su totalidad la documentación es libre, y se necesitan también mínimos conocimientos de UNIX para compilarlo. Su única dependencia es la librería GMP, por lo que sus requisitos son mínimos y ya disponemos de ellos actualmente. Una simple instrucción nos bastaría para compilarlo.

Tras el estudio de estas tres implementaciones, y de algunas otras aún por descubrir, sería necesario su estudio para deducir cuál es la que permite paralelizarse de modo más sencillo sin olvidar cuál es la más potente, evaluar su tolerancia a los fallos o *"bugs"*, etc. En definitiva, si este hito se consiguiera, la capacidad de cómputo que supondría emplear decenas de máquinas ejecutando un algoritmo que puede factorizar por sí solo números de más de 130 sin problemas, (según se puede leer en los sitios web de estos proyectos) podría resultar en un sistema de gran potencia a la hora de conseguir descomponer claves del tamaño que se están empleando actualmente, la verdadera meta final de nuestro sistema.

[50]http://kmgnfs.cti.gr/kmGNFS/Home.html
[51]http://www.math.ttu.edu/~cmonico/software/ggnfs/

# 7.3 RSA@Cloud como sistema multiplataforma

El carácter multiplataforma de los lenguajes empleados (Perl, C, C++, Java) para el desarrollo de los diferentes módulos de los que consta el sistema, sería muy provechoso y relativamente sencillo adaptar el sistema a un entorno independiente de la plataforma y de la arquitectura en el cual el cliente y los servidores se comunicasen entre sí sin importar el sistema operativo de las distintas máquinas (UNIX, Windows, Mac OS).

El módulo **Engine** emplea Perl, C y C++, además de las librerías necesarias para ejecutar los algoritmos codificados en los dos últimos lenguajes, GMP y NTL.

Los sistemas operativos Linux y Mac OS ya llevan Perl integrado y en Windows existen dos posibilidades para instalarlo: Strawberry Perl[52] y ActivePerl[53]. Ambos son gratuitos y se descargan a través de sus respectivos sitios web. Se distribuyen en forma de ejecutables con extensión '.exe' y los scripts pueden ejecutarse a través de la consola de Windows.

Los compiladores de C y C++ son GCC y G++ respectivamente. Al igual que Perl, ambos vienen incluidos en las todas las versiones de Linux y Mac OS. Para Windows existen dos posibles soluciones: Cygwin[54] y MinGW[55]. Ambos se instalan sencillamente a través de archivos de instalación '.exe', por lo que no requiere ningún esfuerzo hacerlo. Es importante que, cuando aparece la opción de incluir el paquete "m4", se selecciona para poder integrar más tarde las dos librerías necesarias para este sistema (GMP y NTL). Al igual que Perl, su manejo para compilar programas se hace desde la consola de Windows.

Las dos librerías especiales para desarrollar programas de cálculo matemático GMP y NTL son los dos últimos componentes necesarios para poder arrancar el módulo Engine. GMP se instala en Mac OS de la misma manera que en los sistemas Linux. Para Windows, una vez instalado CygWin o MinGW su incorporación a GCC y G++ es muy sencilla: ambos tienen la opción de desplegar una consola *"bash"* que funcionan como cualquier terminal de Linux. Escribiendo los mismos comandos que son necesarios para Linux, es posible llevar con éxito su instalación[56].

En cuanto a la instalación de NTL, de nuevo Mac OS funciona de la misma manera que Linux. N cambio, los desarrolladores de NTL disponen en su sitio web de una versión específica para Windows: WinNTL. Las instrucciones también vienen incluidas en el archivo comprimido en el que se presenta, y recomiendan contar con la herramienta de programación Microsoft Visual C++ para hacer este proceso algo más sencillo, el cual está muy bien descrito en su documentación. En caso contrario, el proceso puede ser más arduo, sin embargo, bastaría con compilar el código fuente y crear una librería estática, asegurarse de que el compilador sepa dónde encontrarlos

[52] http://strawberryperl.com/
[53] http://www.activestate.com/activeperl/downloads
[54] http://sourceware.org/cygwin/
[55] http://www.mingw.org/
[56] http://www.cs.nyu.edu/exact/core/gmp/

archivos de NTL (directorio 'include'). Para compilar un programa usando la biblioteca, el compilador debe saber dónde se encuentran ubicados la biblioteca y el directorio de NTL.

El módulo **Forecaster** ya es totalmente multiplataforma, de la misma manera que Java lo es. Para arrancar un ejecutable '.jar' en cualquier sistema operativo únicamente hace falta la máquina virtual de Java JDK, y si se desease hacerlo mediante comandos por consola o terminal, el proceso consta de los mismos pasos independientemente del sistema operativo. En resumen, este módulo es desde ya compatible con cualquier máquina.

**Codeswarm** es básicamente un programa en Java, por lo que podemos atribuir a este módulo lo dicho anteriormente para el módulo anteriormente descrito. Sin embargo, aunque el sistema Mac OS no difiere de Linux en los pasos para su instalación y arranque descritos en el apartado 5.3.2 de este documento, Windows requiere una serie de pasos extra muy particulares, pero que no pasan de añadir tres directorios como variables de entorno del sistema. Los pasos para construir Codeswarm en cualquier plataforma se describen en su repositorio web.[57]

En conclusión, sería relativamente fácil hacer compatible cualquier máquina con nuestro sistema de tal forma que funcionase como máquina cliente del proceso de distribución de tareas en los distintos servidores abriendo un puerto TCP para enviar información por SSH. Para hacer de cualquier máquina un servidor también se necesita activar SSH de entrada y de salida. Mac OS vuelve a no ofrecer variaciones con respecto a Linux al ser un sistema basado en UNIX, y en Windows existe multitud de software libre que permiten la conexión SSH: CygWin incorpora un servidor SSH configurable, FreeSSH Server[58] y KpyM Server[59], entre otros.

En un principio, hasta proceder a una fase de pruebas del sistema para corregir posibles fallos, previsiblemente sobre todo problemas de conexión a máquinas Windows, podría ser factible hacer de nuestro sistema un proyecto multiplataforma a corto plazo.

## 7.4 Ampliación del módulo Forecaster a más proveedores Cloud

Motivado por la amplia oferta en el mercado de computación Cloud, se extenderá la funcionalidad de Forecaster a otros proveedores distintos de Amazon, como pueden ser Windows Azure (Microsoft), Salesforce, Netmagic y Google, entre otros, así como se adaptará la predicción al uso de distintos tipos de máquinas a la vez, favoreciendo la heterogeneidad de la infraestructura y obteniendo mejores resultados para la métrica *C/R*.

[57] http://code.google.com/p/codeswarm/wiki/HowtoBuild
[58] http://www.freesshd.com/
[59] http://www.kpym.com/2/kpym/index.htm

Ante todo, lo primero que se necesita es recopilar información acerca del tipo de instancias y los precios que ofrecen los distintos proveedores de computación en la Nube. El estudio se centrará en algunas de las marcas anteriormente citadas, aunque la oferta en el mercado es muchísimo mayor. Con el objetivo de mostrar todas las alternativas en igualdad de condiciones, nos centraremos en las oferta de pago por demanda, que son el tipo de instancias que se escogieron del servicio Amazon EC2 para realizar este estudio.

**Microsoft Azure**

Microsoft ofrece cinco tipos de instancias que cumplen un patrón muy sencillo, a excepción de las nuevas máquinas de tipo Extra Small.[60] El resto de instancias a partir de la de tipo Small siguen esta regla: el siguiente tipo de instancia más potente tiene el doble de núcleos, el doble de memoria RAM, un poco más del doble de espacio de disco duro y el doble de precio por hora:

| Tipo de Instancia | CPU | Memoria | Almacenamiento | Rendimiento Entrada / Salida | Precio |
|---|---|---|---|---|---|
| Extra Small | 1 GHz | 768 MB | 20 GB | Bajo | $0.05/hora |
| Small | 1.6 GHz | 1.75 GB | 225 GB | Medio | $0.12/hora |
| Medium | 2 x 1.6GHz | 3.5 GB | 490 GB | Alto | $0.24/hora |
| Large | 4 x 1.6GHz | 7 GB | 1,000 GB | Alto | $0.48/hora |
| Extra Large | 8 x 1.6 GHz | 14 GB | 2,040 GB | Alto | $0.96/hora |

Tabla 7.1: Microsoft Azure

**Netmagic**

Este proveedor proveniente de India ofrece una oferta más reducida de instancias, sólo tres, pero no por ello deja de tener precios competitivos, y la posibilidad de elegir entre los sistemas operativos Cent OS[61] y Windows, con opción a muchos otros. Según la FAQ del proveedor, cada CPU es de 2GHz como mínimo:

| Tipo de instancia | CPU | Memoria RAM | Almacenamiento | Precio |
|---|---|---|---|---|
| Small | 2.0 GHz | 1 GB | 20 | $0.13/hora |
| Medium | 2 X 2.0 GHz | 2 GB | 20 | $0.22/hora |
| Large | 4 X 2.0 GHz | 4 GB | 40 | $0.33/hora |

Tabla 7.2: Netmagic. Nota: La Memoria RAM y el almacenamiento son opcionalmente ampliables. Precios convertidos a dólares americanos, originalmente en rupias indias.

Incorporar los datos de éstos y otros proveedores sería muy sencillo de implementar en el módulo Forecaster. En el panel izquierdo de la ventana principal (ver más en

[60] http://www.microsoft.com/windowsazure/offers
[61] http://www.centos.org/

sección 5.3.2) se puede elegir la máquina a elegir, por lo que sería suficiente ampliar el Combo Box de opciones o realizar modificaciones en el panel para seleccionar proveedor de servicios, todo ello a nivel de interfaz. Dentro del paquete de clases 'Maquinas', bastaría con añadir más clases que heredasen de la clase 'Maquina' con la información técnica de las mismas. Para calcular sus funciones de tiempo de ejecución con respecto al tamaño de la clave a descomponer, el proceso es tan sencillo como repetir todos los cálculos realizados con el fin de obtener estas funciones para las cuatro instancias de Amazon EC2 (ver sección 6 del presente documento para más información).

Con esta ampliación del módulo Forecaster, el usuario tendrá la oportunidad de comparar qué proveedor ofrece la mejor relación Coste/Rendimiento.

## 7.5 Ampliación del módulo Forecaster: elección de algoritmo

El módulo de estimación de tiempo de ejecución y coste está diseñado como muestra de lo beneficioso que puede ser el hecho de conocer de antemano los recursos necesarios para computar tareas de factorización para números enteros de gran magnitud. El punto de partida fue dar predicciones en el caso de emplear el algoritmo de división por tentativa y comprobar que realmente se ajusta a los resultados experimentales.

Dar el paso de ampliar esta utilidad al algoritmo de la criba cuadrática sólo requiere contratar una máquina de cada tipo de instancia ofrecida por el proveedor de servicios durante un corto periodo de tiempo y reproducir los cálculos emprendidos para obtener las funciones de tiempo de ejecución para distintos tamaños de clave (ver sección 6).

Incorporar estas funcionalidades al código requerirá modificar la interfaz para dar la opción al usuario de elegir el algoritmo a utilizar en la factorización de la clave introducida, así como nuevas funciones en las clases que heredan de la clase "Maquina", las cuales representan los diferentes tipos de instancia, que devuelvan el tiempo de ejecución en función del tamaño de la clave para la Criba Cuadrática.

Ésta es una mejora que puede hacerse realidad con una pequeña inversión económica, y que podría extenderse a cualquier algoritmo que se implementase para el módulo Engine, incluyendo el algoritmo GNFS, tal y como se ha propuesto para un trabajo futuro en un apartado superior de esta sección.

## 7.6 Selección de puerto para el establecimiento de conexiones SSH

Durante las pruebas a las que sometimos el módulo Engine para testar su correcto funcionamiento, se observó la posibilidad de aumentar el número de máquinas que actuaran como servidor empleando todas los equipos existentes en las redes locales creadas por los routers que conectaban los servidores a la red.

Por ejemplo, si una máquina cliente se conectaba a dos IP para ser empleadas como servidores, ésta empleaba el puerto 22 por defecto en cada una de ellas. Si el sistema fuera capaz de elegir una máquina por su nombre de usuario, IP y por el puerto de conexión, podría establecer conexión con diferentes máquinas con una misma IP pública, empleando puertos diferentes para las comunicación entre máquinas a través de SSH.

Esta conexión múltiple a varias máquinas en una misma dirección sería un gran avance para enlazar servidores particulares con el fin de aumentar los recursos físicos disponibles para la factorización de claves, en el caso de querer emplear equipos domésticos conectados a la Red y así ahorrar costes. El grupo ha investigado sobre esta nueva funcionalidad y es perfectamente posible si se consiguen solucionar los problemas encontrados para establecer la conexión automática bidireccional en el script 'conexión.pl'. El comando "sshpass" (ver sección 5.3) no ofrece opción de elegir puerto y, en las pruebas realizadas, las instrucciones a través de este componente no se efectuaban correctamente aunque incluyeran opción de elegir puerto (como "ssh"). Este aspecto es un tema en el que sería interesante profundizar, aunque sólo fuese con la intención de continuar el aprendizaje acerca de las conexiones remotas a través de SSH.

## 7.7 Conexión con máquinas en la misma LAN a través de red WAN

Una forma de sacar más partido a las características del Engine será teniendo la posibilidad de alcanzar máquinas tanto en la propia red local del cliente como varias máquinas en una misma red local remota. La arquitectura actual no permite alcanzar máquinas en la misma LAN[62] del cliente ni varios servidores en la misma red local remota, por lo que sería conveniente investigar sobre esta nueva posibilidad de obtener más recursos computaciones para factorizar claves en un menor tiempo.

[62] Local Area Network – Red de área local.

# 7.8 Uso de las nuevas instancias Cluster GPU para acelerar el proceso de factorización

Existe una familia de tipos de máquina en Amazon EC2 pensada para aquel que demande una mayor capacidad de cálculo: las instancias de Computación en Cluster (CC) y las de Clusters de GPUs (CGPU).[63] Estas instancias, diseñadas para aplicaciones de Computación de Alto Rendimiento tienen las siguientes características:

- Cuando se ejecuta un cluster de instancias, existe poca latencia entre ellas. Además el ancho de banda entre máquinas pertenecientes al cluster es de 10 Gbps y el cluster puede estar formado por más de 128 máquinas.
- Tanto CC como CGPU especifican la arquitectura del procesador para que los desarrolladores puedan optimizar sus aplicaciones para dichas arquitecturas.

En la siguiente tabla se pueden ver las prestaciones de cada uno de ellos:

| CPU | GPU |
|---|---|
| 23 GB de memoria<br>33.5 unidades de computación<br>(2 x Intel Xeon x5570 quad-core“Nehalem”)<br>1690 GB de almacenamiento<br>Plataforma de 64 bit<br>E/S: 10 Gigabit Ethernet<br>Nombre API: cc1.4xlarge | 22 GB de memoria<br>33.5 unidades de computación<br>(2 x Intel Xeon x5570 quad-core“Nehalem”)<br>2 x ENVIDIA Tesla “Fermi” M2050 GPUs<br>1690 GB de almacenamiento<br>Plataforma de 64 bit<br>E/S: 10 Gigabit Ethernet<br>Nombre API: cg1.4xlarge |

Tabla 7.3: Cluster CPU Vs cluster GPU

Ambas están disponibles en la región de Virginia del Norte sólo para sistemas Linux.

Como se puede ver en la tabla, las instancias GPU cuentan con dos tarjetas gráficas ENVIDIA. En la actualidad se sabe que las tarjetas gráficas son el acelerador hardware óptimo para comprometer claves. Además, sus 33.5 unidades de cálculo, muy por encima del resto de tipos de máquina de Amazon, conseguirían reducir el tiempo de procesamiento de una tarea drásticamente, para cualquier algoritmo de factorización, ya sean los implementados en este proyecto u otras futuras implementaciones, como puede ser GNFS.

[63] http://aws.amazon.com/hpc-applications/#HPCEC2

## 7.9 Recuperación dinámica ante errores en el módulo Engine

En el estado actual de desarrollo del módulo Engine, el sistema funciona perfectamente siempre que alguna de las máquinas remotas que actúan como servidores no sufran ningún percance. Esto quiere decir que si un equipo pierde temporalmente la conexión a la red, queda totalmente inutilizable debido a algún fallo en el sistema operativo o en disco duro, o cualquier otra causa, existe gran probabilidades de que la máquina cliente no sepa reaccionar a este imprevisto y entre en un bucle infinito intentando conectar de nuevo o cualquier otro tipo de *"bug"*.

Sería conveniente estudiar la posibilidad de incorporar mejoras que incorporasen un mecanismo de verificación que comprobase la correcta finalización de cada instrucción a una máquina remota, como obligar a dicha máquina a enviar una confirmación con el resultado de la ejecución, como si de un ACK en el protocolo TCP[64] se tratase. Si la máquina remota estuviera computando una subtarea en el momento de quedarse fuera de servicio, será imprescindible que la máquina cliente activara un contador al enviar la información de la subtarea. En el caso de que el contador llegara a un tiempo límite máximo para la ejecución de cada subtarea fijado de antemano, ésta tendría que dar al servidor por perdido y reenviar la subtarea a otra máquina. De esta manera la máquina cliente podría actuar frente a situaciones imprevistas y descartar la máquina afectada de la lista de servidores disponibles en la ejecución del módulo, impidiendo su detención y el consecuente gasto de tiempo y dinero que eso supondría.

Otra solución más sofisticada permitiría la inclusión de las máquinas afectadas por algún imprevisto en la lista de máquinas disponibles una vez que volvieran a dar una señal que indicara que volvieran a estar disponibles, o incluso añadir otras máquinas sobre la marcha sin alterar la ejecución del sistema.

La recuperación en caliente ante errores es claramente una prioridad y sería muy importante conseguir este objetivo pensando en la descomposición de factores que requieran un gran número de máquinas, con el fin de garantizar la inversión requerida y la viabilidad del proyecto.

## 7.10 Interfaz gráfica para el módulo Engine e integración con Forecaster

Para mayor comodidad y eficiencia de uso, sería un acierto implementar una sencilla interfaz gráfica que permitiera introducir y generar los parámetros iniciales del módulo Engine. La idea consiste en desarrollar un sistema de ventanas donde se pudieran elegir las siguientes opciones:

[64] http://www.faqs.org/rfcs/rfc793.html

- Tipo de algoritmo
- Clave
- Número de subtareas
- Lista de direcciones IP, nombres de usuario, contraseña y puerto

Situándonos al nivel del sistema de archivos, el programa creará y modificará todos los ficheros de texto de la carpeta 'datos', donde se generan los archivos 'algoritmo.txt', 'clave.txt', 'rangos.txt' y 'maquinas.txt', que actualmente se generan manualmente o ejecutando scripts Perl por terminal (ver sección 5.3). A continuación, la interfaz debería disponer de dos botones que permitieran realizar la conexión automática con los servidores y comenzar con el reparto y el cómputo de las subtareas respectivamente.

Para que el módulo Engine se integrase plenamente con el módulo Forecaster, el sistema arrancaría al comienzo este último. Una vez realizadas las estimaciones convenientes y con los resultados a la vista del usuario, se tomarían estos datos para ser los parámetros iniciales del sistema de ventanas ideado para el Engine. Antes de proceder a la conexión y ejecución, el usuario tendría la oportunidad de modificar estos parámetros (algoritmo, clave, número de subtareas, lista de IPs) si finalmente no desea emplear la configuración sugerida por el Forecaster. A su vez, tampoco supondría mucho más esfuerzo incluir la opción de arrancar Codeswarm para visualizar y grabar en formato de video una secuencia generada a partir del archivo log en formato XML creado en la carpeta 'resultados' tras haber finalizado la ejecución del módulo Engine.

En principio, Java parece ser de nuevo la herramienta ideal para este objetivo, ya que el sistema unificado en una sola aplicación Java lo haría multiplataforma. Desarrollar un instalador para las plataformas Linux, Mac OS y Windows en el que se instalasen todas las librerías necesarias y los tres módulos del sistema completaría el desarrollo de una aplicación unificada.

## 7.11 Desarrollo de sistema Engine-Forecaster-Codeswarm combinado con el API de Amazon

Sin duda, este sería uno de los proyectos que se podrían desarrollar a partir de este sistema. En su sitio web, Amazon ofrece multitud de APIs para programadores para gran parte de sus servicios en la Nube. Para el caso de Amazon EC2 existe una API muy interesante llamada Amazon EC2 API Tools.[65] Este paquete *software* permite instalar ciertos componentes en el sistema operativo de cualquier equipo que habitan al usuario poder administrar su cuenta en Amazon EC2, así como las instancias que tiene reservadas o en funcionamiento, desde la terminal de comandos, en lugar de usar un explorador web para conectarse e identificarse en la página habilitado para ello por Amazon.[66]

Una vez instalada la API en el equipo, podría usarse la misma para desarrollar una aplicación con una interfaz gráfica sencilla que permitiera administrar una cuenta en

[65] http://aws.amazon.com/developertools/351
[66] http://docs.amazonwebservices.com/AWSEC2/latest/GettingStartedGuide

EC2 y ejecutar al menos las operaciones más comunes. La GUI debería contener formularios que generasen los comandos que llevan a cabo la acción deseada al clicar un botón de la misma, por poner un ejemplo.

Para llevar esto a cabo, es necesario estudiar la guía de comandos de esta API que ofrece Amazon en su web, aunque antes se debería optar por dominar la instalación de este paquete *software*, la cual es bastante compleja. En este caso, Amazon apenas documenta los pasos a seguir, pero se pueden encontrar valiosos tutoriales en algunos foros de internet, como este[67] o este otro[68]. Se requiere la máquina virtual de Java para llevar la instalación a buen término, y puede hacerse en cualquier SO, aunque algunos plantean más facilidades que otros. Es más, incluso podría programarse un script Perl que lo hiciera automáticamente para evitar al usuario hacerlo manualmente.

Como se ha señalado, éste es un proyecto de mayores dimensiones, sobre todo por el hecho de tener que integrar todas y cada una de las instrucciones por línea de comandos que ofrece esta API con una GUI de una aplicación Java, pero se conocen las ideas generales para desarrollar esta funcionalidad.

[67] https://help.ubuntu.com/community/EC2StartersGuide
[68] http://blog.bottomlessinc.com/2010/12/installing-the-amazon-ec2-command-line-tools-to-launch-persistent-instances/

# 8. Conclusiones

Durante las etapas de diseño, planificación, desarrollo y análisis del sistema RSA@Cloud, cada progreso en este proyecto ha supuesto el descubrimiento de ramas del mundo de la informática que el grupo de trabajo no conocía hasta ahora, así como el dominio de lenguajes y herramientas nuevos, la obtención de una visión más completa del estado del arte actual de las tecnologías que han sido empleadas, incluyendo su proyección en un futuro próximo, e incluso mayores conocimientos en ciertos campos en un principio aparentemente ajenos a nuestro campo de trabajo pero en realidad directamente relacionados con las ciencias de la información como la Criptografía.

En esta última sección se enumeran los diferentes aspectos que engloban las bases que se han establecido tras la ejecución de este proyecto, en relación a:

- Las tecnologías utilizadas para llevarlo a término, como lo han sido:
  - La computación Cloud.
  - La programación paralela.
  - El criptosistema RSA.

- Los conocimientos adquiridos en las diferentes herramientas que han sido útiles en la construcción de los diferentes módulos que conforman el proyecto RSA@Cloud.

- Las características y capacidades del sistema, así como su potencial.

**Cloud Computing**

La computación Cloud es el paradigma de la computación distribuida que reúne las mejores virtudes de las tecnologías desarrolladas sobre este campo, tales como los modelos de clúster, Intranet Computing, Internet Computing, P2P y Grid [YONG]. Esta nueva tecnología, gracias al desarrollo de la virtualización, la arquitectura de red orientada a servicios e Internet, permite un alto grado de rendimiento y disponibilidad, balanceo de carga y escalabilidad así como elasticidad, lo que permite personalizar la cantidad de recursos disponibles rápidamente, dando una sensación de acceso por demanda a recursos de computación infinitos. Un rendimiento medio mayor, unido a la posible portabilidad de la información y el ahorro que supone no abordar gastos de mantenimiento y actualización son otras cualidades que posicionan esta tecnología a la vanguardia de la computación distribuida.

En la actualidad, el criptoanálisis de un sistema RSA sigue requiriendo enormes cantidades de potencia computacional. La computación Cloud nos abre una nueva ventana hacia la factorización eficaz ofreciendo acceso a dicha capacidad de cálculo bajo demanda del usuario, dándole la posibilidad de saber en cada momento el precio de la potencia demandada.

**Programación paralela**

El uso de la computación Cloud para la paralelización de tareas, supone grandes ventajas y mejoras, sobre todo en función del tiempo, ya que permite a cualquier usuario un mayor acceso de capacidad de computación. En nuestro sistema se puede observar como la factorización de números grandes, en concreto de claves RSA, conlleva demasiado tiempo.

Este tiempo se reduce de forma considerable con la transformación de un programa que trabaja de forma secuencial tareas independientes en una sola máquina a un sistema parelizable mediante el cual se puede dividir un trabajo computacional en subtareas ejecutables en diferentes equipos comunicados, que juntas proporcionan el mismo resultado final en un tiempo menor. Dicho tiempo se estima que sea inversamente proporcional al número de máquinas empleadas en la paralelización de la tarea.

**Panorama actual de las claves RSA**

Nuestro sistema puede ser utilizado para evaluar la seguridad de una clave RSA determinando el coste necesario para comprometer su resistencia. Precisamente, uno de los objetivos de RSA@Cloud es proporcionar una visión del potencial de las tecnologías de la computación y su velocidad de cálculo, de la infraestructura de la Red y del panorama actual de la matemática orientada a la criptografía.

Las estimaciones de algunos criptógrafos a lo largo de las décadas han quedado frecuentemente en papel mojado al no prever la rápida evolución de los avances científicos y tecnológicos, como les ocurrió a Lenstra y Manasse en 1997 (ver sección 4.6). Los concursos propuestos por RSA han servido para este propósito y, ahora que esta empresa declara haberlos suprimido conocer el estado actual del campo de la Criptografía[69], RSA@Cloud puede servir como base para establecer una herramienta que mida las capacidades de algoritmos ya existentes y futuros y convertirse en un componente a tener en cuenta para las auditorías de seguridad de claves RSA.

**Conocimientos adquiridos**

La implementación de un sistema de estas características ha supuesto el aprendizaje y perfección de ciertas habilidades que incluyen: la búsqueda selectiva de información e investigación en la Red, artículos y publicaciones, diseño y planificación de proyectos, reparto de tareas y trabajo en equipo, además de diferentes conocimientos acerca del estado del arte de los campos de investigación que explora este proyecto, nociones de sistemas operativos y técnicas de programación como:

- Paradigmas de los sistemas distribuidos, en especial claro está, la computación Cloud.
- Iniciación en la programación paralela con el fin de aprovechar arquitecturas de computación multinúcleo y sistemas distribuidos.
- Conocimientos criptográficos, más concretamente en el campo de la criptografía de clave pública, con RSA como ejemplo.
- Dominio de sistema de ficheros y terminal de comandos en sistemas

[69] http://www.rsa.com/rsalabs/node.asp?id=2092

UNIX/Linux y Windows.
- Programación de scripts Perl con invocaciones al sistema, uso de protocolo SSH para comunicación en red, instalación de componentes y librerías, programación en C, C++, Java y MathLab.
- Iniciación a la investigación científica mediante la publicación de un artículo científico y redacción de una memoria de proyecto a nivel académico.

**Características y capacidades de RSA@Cloud**

Este sistema ha conseguido alcanzar los siguientes objetivos:

- La creación de un módulo que aproveche las virtudes de la computación en la Nube para distribuir algoritmos de factorización de enteros en los que se han aplicado las propiedades de la programación paralela con el fin de atacar módulos RSA.

- El desarrollo de una aplicación que sienta las bases para obtener sin gasto económico alguno una estimación veraz del tiempo total de ejecución y coste que suponen descifrar una clave RSA sobre una red de máquinas en la Nube.

- Representar visualmente de forma atractiva y comprensible la comunicación entre máquinas y la ejecución del sistema: conexiones entre máquinas y transferencia de archivos.

- Implementar un sistema modular con un gran potencial de mejora en el cual se puedan ajustar sus diferentes componentes a la inclusión de nuevas implementaciones de algoritmos o los últimos modelos de instancias en la Nube, prolongándose así la vida útil del proyecto.

- Se han conseguido hitos como descifrar claves de 50 dígitos en menos de 3 horas con sólo 8 máquinas disponibles con el algoritmo de la criba cuadrática, una mínima muestra del verdadero potencial de RSA@Cloud.

# Referencias

**(en orden alfabético)**

**[ACRI]** *Aplicaciones Criptográfica*s, Segunda Edición (Junio 1999): tercer libro. Dpto. de Publicaciones de la Escuela Universitaria de Informática de la Universidad Politécnica de Madrid, España. ISBN 83-87238-57-2.

**[ALCO]** Alberto Alcocer, *Cloud: Tipos de nube.* Junio 2010.

**[ARMB]** Michael Armbrust, Armando Fox, Rean Griffith, Anthony D. Joseph, Randy Katz, Andy Konwinski, Gunho Lee, David Patterson, Ariel Rabkin, Ion Stoica, y Matei Zaharia, *A view of Cloud Computing Communications of the ACM*, vol. 5 3 no. 4, Abril 2010.

**[BARR]** Daniel J. Barrett, Richard Silverman, *SSH, The Secure Shell: The Definitive Guide.* Capítulo 2.5. O'Reilly, Enero 2001. ISBN: 0-596-00011-1.

**[BEZO]** Ann Byers, *Jeff Bezos: The Founder of Amazon.com (Internet Career Bios)*. ISBN: 1404207171.

**[BOGG]** Profesor-Juan Boggio. Integrantes- Erika de Pablos Vélez Espinosa, Ana Laura Rodríguez García, Manuel Plot, Benjamín Palacios Díaz Juan Carlos González*,"Caso Amazon VS. Barnes&Noble" Planeación estratégica*. 2010. http://www.buenastareas.com/ensayos/Caso-Amazon/296552.html

**[BRIG]** Matthew E. Briggs, *An introduction to the General Number Field Sieve*. Faculty of the Virginia Polytechnic Institute and State University, Abril 1998.

**[BUCH]** Johannes Buchmann, *Introduction to Cryptography, Second Edition*. ISBN: 978-0-387-21156-5

**[BUYY]** R. Buyya, C. Shin Yeo, S. Venugopal, J. Broberg & I. Brandic, *Cloud computing and energing IT plataforms: Vision, hype, and reality for delivering computin as the 5th utility*, Future Genetarion Computer Systems vol. 25, Issue 6, Páginas. 599-616, Junio 2009.

**[CABA]** Pino Caballero Gil, *Algunos hitos de la criptografía del siglo XX.* Revista Números, 43-44. Páginas 405-408, SEP-DIC 2000.

**[CARR]** Nikolas Carr, *The Big SwitchRewiring the Worldfrom Edison to Google.* ISBN: 0393062287.

**[CHIN]** Laurence Tianruo Yang, Li Xu, and Man Lin, *Integer factorization by a Parallel GNFS Algorithm for Public Key Cryptosystems*. L.T. Yang et al. (Eds.): ICESS 2005, LNCS 3820, Páginas 683–695, 2005.

**[CINC]** Cinco días, *Los servicios de "Cloud Computing" crecerán un 16% en 2010.* Junio 2010. http://www.cincodias.com/articulo/empresas/servicios-cloud-computing-creceran-2010/20100625cdscdiemp_32/

**[CLTV]** Cloudtweaks, *Cloud TV may replace local TV altogether*... 8 de Diciembre, 2010. http://www.cloudtweaks.com/2010/12/cloud-tv-may-replace-local-tv-altogether

**[DAMI]** Marcelo Damián Parrino, *Análisis de Rendimiento para Soluciones de Cloud Computing*. Octubre 2010.

**[DEVE]** A.K. Lenstra, H.W. Lenstra, Jr. *The development of the Number Field Sieve*. Springer-Verlag Berlin Heidelberg New York. ISBN 3-540-57013-6.

**[DIRE]** Whitfield Diffie, Martin E. Hellman, *New Directions in Cryptography*. Invited Paper, 1976.

**[DUMM]** Judith Hurwitz, Robin Bloor, Marcia Kaufman, Fern Halper, *Cloud Computing for Dummies (Cap. 1)*, ISBN: 978-0-470-48470-8.

**[ELLE]** Jr. H. W. Lenstra, *Elliptic curve factorization, personal communication via*. Samuel Wagstaff Jr., 1985.

**[FIGU]** R. Figueiredo, P.A. Dinda, J. Fortes, *Guest Editors' Introduction: Resource Virtualization Renaissance*, Computer, vol.38, no.5, Páginas 28- 31, Mayo 2005.

**[FISH]** Sharon Fisher, *NetEX HyperIP Accelerates Data Transfer*. 9 Abril 2010

**[FUCR]** Henk C.A. van Tilborg, *Fundamentals of Cryptology. A Profesional Reference and Interactive Tutorial.* ISBN: 0-306-47053-5. 2002.

**[HURW]** Judith Hurwitz, Robin Bloor, Marcia Kaufman, Fern Halper, *Cloud Computing for Dummies (Cap.2)*.ISBN:978-0-470-48470-8.

**[INFA]** Arjen K. Lenstra, *Integer Factoring*. Designs, Codes and Cryptography, 19, 101-128, 2000.

**[INTE]** *Internet2 Community and the large hadron collider*. http://www.internet2.edu/pubs/LHC-infosheet.pdf

**[IVOR]** Carlos Ivorra Castillo, *Álgebra*. Universidad de Valencia, 2010. http://www.uv.es/ivorra/Libros/Algebra.pdf

**[JAKL]** Michael Jakl, University of Technology Vienna, *Representational State Transfer*. Mayo 2000. http://www.w3.org/TR/2000/NOTE-SOAP-20000508/

**[JENS]** Leslie Jensen, Master Thesis, *Integer Factorization.* Department of Computer Science, Universidad de Copenhague. Otoño 2005.

**[JUVE]** G. Juve, E. Deelman, K. Vahi, G. Mehta, B. Berriman, B.P. Berman, P. Maechling, *Scientific workflow applications on Amazon EC2*. Workshop on Cloud-based Services and Applications in Conjunction with 5th IEEE International Conference on e-Science, e-Science‟09, 2009.

**[KIMI]** Ian Kiming, *The ψ-funcion and the complexity of Dixon's factoring algorithm.* Department of Mathematics, University of Copenhagen.

**[LAND]** Eric Lindquist, *The Quadratic Sieve Factoring Algorithm*. Math 488: Cryptographic Algorithms, Diciembre 2001.

**[LUCE]** M.J. Lucena López, *Criptografía y seguridad en computadores*. versión 4-0.7.51. Universidad de Jaén, Junio 2008.

**[MAGU]** James Maguire, Jeff Vance, Cynthia Harvey. *85 Cloud Computing Vendors Shaping the Emerging Cloud*. Agosto 2009. http://itmanagement.earthweb.com/features/article.php/12297_3835941_2/85-Cloud-Computing-Vendors-Shaping-the-Emerging-Cloud.htm

**[MART]** Xabier Martorell, *Cloud Computing: Antecedentes históricos*. 14 Febrero 2010. http://xmartorell.wordpress.com/2010/01/14/cloud-computing-la-historia-ya-esta-acostumbrada

**[MELL]** Peter Mell, Timothy Grance, *NIST Computer Security Division & Computer Security Resource Center.* Special Publication 800-145, Enero 2011.

**[MOBR]** Morrison y John Brillhart, *A method of factoring and the factorization of the F7.M.A.*Mathematics of Computation, vol. 29, no. 129, Páginas 183-205, Enero 1975.

**[MOHA]** Arif Mohamed, *A story of Cloud Computing*. 27 Marzo 2009. http://www.computerweekly.com/Articles/2009/06/10/235429/A-history-of-cloud-computing.htm

**[MURP]** Brian Antony Murphy, *Polynomial selection for the number field sieve integer factorisation algorithm*. PhD - thesis (1999).

**[PACE]** Ariel Pacetti, *Teoría de números*, capítulo 2.1. Notas del curso para las Jornadas de Criptografía y Códigos Autocorrectores. Universidad Nacional de Mar del Plata, Noviembre 2006.

**[POLE]** J.L. Vázquez-Poletti, Eduardo Huedo Cuesta, Rubén Santiago Montero, Ignacio Martín Llorente, *Una visión global de la tecnología Grid*. Departamento de Arquitectura de Computadores y Automática Universidad Complutense de Madrid, Laboratorio de Computación Avanzada, Simulación y Aplicaciones Telemáticas y Centro de Astrobiología. (CSIC-INTA)

**[POME]** Carl Pomerance, *A Tale of Two Sieves*. Diciembre 1996.

**[RAGH]** Kaushik Raghupathi, *5 Key Events in the history of Cloud Computing*. Febrero 2011. http://architects.dzone.com/news/5-key-events-history-cloud?utm_source=feedburner&utm_medium=feed&utm_campaign=Feed%3A+zones%2Fdotnet+%28.NET+Zone%29

**[RAVI]** Barath Raghavan, Kashi Vishwanath, Sriram Ramabhadran, Kenneth Yocum, y Alex C. Snoeren, *Cloud Control with Distributed Rate Limiting*, SIGCOMM'07, Kyoto, Japan, Agosto 2007. http://developer.att.com/developer/tierNpage.jsp?passedItemId=200146

**[RSCH]** Randal L. Schwartz, Tom Phoenix, *Learning Perl*, tercera edición. O'Reilly, Julio 2001. ISBN: 0-596-00132-0.

**[RUSS]** Gregory V. Bard. *Accelerating Cryptanalysis with the Method of Four Russians*. Departament of Applied Mathematics and Scientific Computation, Julio 2006.

**[SAAV]** Esteban Saavedra López, *Cloud Computing.* CEO Opentelematics, Internacional Bolivia.

**[SASA]** Sasa Mrdovic, Branislava Perunicic. *Kerckhoffs' Principle for Intrusion Detection.* 7 Marzo 2008.

**[SCHU]** Lutz Schubert, *The Future of Cloud Computing Opportunities for European Cloud Computing Beyond 2010*. Public Version 1.0, 2010.

**[SHOU]** Victor Shoup, *A Computational Introduction to Number Theory and Algebra, version 2*. Junio 2008.

**[SHRO]** Shane Robinson, *EaaS: A blue sky view of the cloud*. http://www.hp.com/hpinfo/initiatives/eaas/SR_EaaS_viewpoint.pdf

**[SILV]** Robert D. Silverman, *The Multiple Polynomial Quadratic Sieve*. Mathematics of Computation, vol. 48, no. 177, Páginas 329-339. Enero 1987.

**[SMAR]** Steven W. Teppler. *Smart Card System and methods for proving dates in digital files.* Patente número US 6.792.536 B1, Septiembre 2004.

**[SMPO]** Carl Pomerance, *Smooth numbers and the quadratic sieve.* MSRI Publications vol. 44, 2008.

**[SPRI]** *Introduction to Cryptography, principles and applications, Second Edition*. Página 61. Springer Berlin Heidelberg New York. ISBN-13 978-3-540-49243-6.

**[SUNG]** *Sungard Cloud Computing*, SunGard Availability Services. 2010.

**[SYAN]** Song Y. Yan, *Cryptanalytic Attacks on RSA*. Springer Science+Business Media, LLC. e-ISBN-13: 978-0-387-48742-7.

**[TAYL]** Guillermo Taylor, *Definición de Cloud Computing*, 25 Agosto 2010. http://blogs.technet.com/b/guillermotaylor/archive/2010/08/25/definici-243-n-de-cloud-computing-por-el-nist.aspx

**[TELA]** Ishay Haviv, *Attack on RSA with Low Public Exponent*. Lattices in Computer Science, Universidad de TelAviv, 2004.

**[TNFS]** A.K. Lenstra, H.W. Lenstra, Jr., M.S. Manasse, J.M.Pollard, *The Number Field Sieve*. 1990.

**[TORK]** Nathan Torkington, *Perl Cookbook*. Publicado por O'Reilly, Agosto 1998. ISBN 1-56592-243-3.

**[TUNI]** John M. Kowalik, *Alan M. Tuning*. 1995

**[UJAL]** Manuel Ujaldón Martínez, *Arquitectura del PC*. Edición 2003. Volumen I. Microprocesadores. Capítulo 1.4. ISBN: 84-95391-86-4.

**[URQU]** James Urquhart, *A better way to understand Cloud Computing*. January 28. http://news.cnet.com/8301-19413_3-10152106-240.html

**[VAPO]** J.L. Vázquez-Poletti, G. Barderas, I.M. Llorente, P. Romero, *A Model for Efficient Onboard Actualization of an Instrumental Cyclogram for the Mars MetNet Mission on a Public Cloud Infrastructure*. In Proc. PARA2010: State of the Art in Scientific and Parallel Computing, Reykjavik (Iceland), June 2010, Lecture Notes in Computer Science, Volume in press, 2011.

**[WIED]** Douglas H. Wiedemann (Member IEEE), *Solving Sparse Linear Equations Over Finite Fields.*IEEE Transactions on Information Theory, Vol. IT-32, Nº1, Enero 1986.

**[WALL]** Larry Wall, Tom Christiansen, Jon Orwant, *Programming Perl*, tercera edición. O'Reilly, Julio 2000. ISBN: 0-596-00027-8.

**[YYAN]** Song Y.Yan*, Primality Testing and Integer Factorization in Public-Key Cryptography*, Second Edition. ISBN-13: 978-0-387-77267-7.

**[YONG]** I. Foster, Yong Zhao, I. Raicu, S. Lu, *Cloud Computing and Grid Computing 360-Degree Compared*, Grid Computing Environments Workshop, GCE '08. Páginas 1-10, Noviembre 2008.

# Índice de Figuras

# Índice de Tablas

# Apéndice I: Implementación de los algoritmos de factorización

**Algoritmo de la división por tentativa**

El algoritmo de la división por tentativa ha sido implementado en C con la biblioteca GMP, gracias a la cual conseguimos la precisión adecuada para números grandes.

Éste es el algoritmo más simple de factorización. Como ya vimos en la sección 3.6.1 anteriormente, probamos a dividir todos los primos desde el primero, el dos, hasta la raíz cuadrada del número a factorizar. Como en nuestra versión no disponemos de una tabla de números primos (ya que no permitiría la paralelización del algoritmo en tareas), probamos el primer primo, esto es, el dos. Con él descartamos todos los números pares, por lo que después de ello probamos con el tres y a partir de este, vamos sumando dos para probar a dividir con todos los números impares. Muchos números impares no son primos, pero esta es una forma fácil e intuitiva de resolver el problema. Si introdujésemos una comprobación de si es número primo o no la complejidad del algoritmo empeoraría.

Una vez terminada la búsqueda, hayamos encontrado o no el factor, creamos un archivo de texto con la información detallada de la ejecución, con el siguiente formato:

- La primera línea indica si se ha encontrado o no:

> **No** se ha encontrado el factor en el rango indicado *$límiteInferior $límiteSuperior*
> ó
> **Sí** se ha encontrado un factor de n. Es *$factor*
> …

- El resto son los valores de la ejecución:

> …
> El valor de n es *$numero*
> El valor del límite inferior es *$limiteInferior*
> El valor del límite superior es *$limiteSuperior*
> El valor de sqrt_n es *$raizCuadrada.*

Donde:

a. *$límiteInferior* es el límite inferior del intervalo.
b. *$límiteSuperior* es el límite superior del intervalo.
c. *$factor* es el factor encontrado.
d. *$numero* es el número a factorizar.
e. *$raizCuadrada* es la raíz cuadrada del número a factorizar, máximo valor de búsqueda.

**Algoritmo de Criba Cuadrática**

El algoritmo de Criba Cuadrática ha sido implementado con NTL integrado con GMP para la máxima precisión de los números. Como ya vimos en la sección 4.6.3, el algoritmo está dividido en cuatro fases. Sin embargo, el técnica de cribado es el proceso más pesado dentro del algoritmo, por lo que éste es dividido en dos etapas:

- Proceso de criba.
- Creación y reducción de la matriz de resultados.

Vamos a ver ambos por partes. El proceso de criba abarca las dos primeras fases del algoritmo:

1. *Configuración de la base de factores y del intervalo de criba.*

Calculamos los parámetros referentes al tamaño de la base de factores *'B'* y al límite del intervalo de criba *'M'* atendiendo a las siguientes fórmulas recomendadas por Eric Landquist en [LAND]:

$$B = \left(e^{\sqrt{\ln(n)\ln(\ln(n))}}\right)^{\sqrt{2}/4}$$

$$M = \left(e^{\sqrt{\ln(n)\ln(\ln(n))}}\right)^{3\times\sqrt{2}/4}$$

Después, calculamos qué números primos son válidos para la base de factores. Recordemos que son aquellos cuyo Símbolo de Legendre es igual a uno. En NTL existen dos funciones que nos ayudan a este proceso, cuya sintaxis NTL es:

- long Jacobi(const ZZ& a, const ZZ& n);

Calcula el Símbolo de Jacobi de *'a'* y *'n'*, asumiendo que 0 <= a < n, con *'n'* impar.

- void NextPrime(ZZ& n, const ZZ& m, long NumTrials=10);

Devuelve en *'n'* el siguiente primo más pequeño mayor o igual que *'m'*.

Posteriormente inicializamos el intervalo de criba, que está implementado por un vector de 'elementoLog', un tipo de datos creado expresamente para ello que contiene:

a. Punto del intervalo de criba '$x_i$'.
b. Valor de la función cuadrática $f(x_i)$.
c. Vector de exponentes. Cada elemento de este vector es un elemento tipo 'par', el cual representa la pareja <número_primo, exponente>, es decir, el exponente al que se eleva el primo dentro de la factorización de $f(x_i)$.

Por último, como la versión de la Criba Cuadrática está implementada con logaritmos (ver sección 4.6.3, apartado 2.2), calculamos el valor de la cota a partir de la cual se comprobará si factoriza un elemento del intervalo de criba. Este valor se calcula dependiendo del tamaño de la clave a factorizar, teniendo:

| Si el tamaño es menor de 25 dígitos: | Si es mayor: |
|---|---|
| $\frac{1}{2}\ln(n) + \ln(M) - T\ln(p_{max})$. | $\log(\frac{M * \sqrt{\frac{n}{2}}}{pmax^T})$ |

Extraídas de forma empírica en [SILV] y aquí.[70]

2. *Proceso de criba.*

Sabemos que:

- $Si\ p\ es\ un\ factor\ primo\ de\ f(x_i), entonces\ p \mid f(x_i + k \times p), k \in \mathbb{Z}$ [1].

Para cada factor de la base de factores se calcula la congruencia:

- $f(x_i) = \left(x_i + \lfloor\sqrt{n}\rfloor\right)^2 - n \equiv 0\ (mod\ p) \rightarrow (x_i + \lfloor\sqrt{n}\rfloor)^2 \equiv n(mod\ p)$.

Y se procede a cribar el intervalo asociado a la tarea que se esté procesando según la progresión aritmética de las soluciones. Como hemos implementado la segunda versión, hemos inicializado el intervalo de criba inicialmente a cero y cada elemento que cumple la propiedad [1] se le suma el número de bits del factor con el que se cumple dicha progresión aritmética. Una vez procesados todos los elementos de la base de factores comprobamos qué elementos superan la cota *'T'* calculada en el paso 1. Estos son los elementos más probables de factorizar completamente sobre la base de factores, así que lo comprobamos mediante división por tentativa, calculando para cada primo que lo divida la mayor potencia que lo divide.

Por último, la salida del algoritmo es un archivo de texto en el que se recogen todos los elementos del intervalo de criba que factorizan completamente sobre la base de factores. Cada elemento muestra el valor dentro del intervalo, el valor de la función cuadrática una vez factorizado (todos los resultados válidos tienen este valor a uno) y una lista de pares primo-exponente con la factorización de la función cuadrática. Por ejemplo, un posible elemento sería:

```
xi: -254
F(xi): 1
Parejas (basePrimo, exponente):
(2, 1)
(3, 1)
(17, 1)
(29, 2)
(-1, 1)
```

Como podemos comprobar, para este ejemplo $n = 87463$:

[70] http://www.comp.nus.edu.sg/~jiezhi/cs3211/

$$f(x_i) = \left(x_i - \sqrt{n}\right)^2 - n = 41 * 41 - 87463 = -85782$$

$$-85782 = 2 * 3 * 17 * 29^2 * (-1).$$

La segunda parte del algoritmo comprende las dos últimas etapas. Esta parte se inicia en la máquina cliente, lanzada por el monitor una vez que se han recopilado los resultados necesarios para tener garantía de éxito. El módulo de álgebra lineal ha sido implementado en C con GMP, y funciona de la siguiente manera:

El módulo de álgebra lineal recibe un archivo de resultados generado, con los parámetros de la configuración de la criba cuadrática y los resultados recogidos en las distintas tareas distribuidas. El archivo tiene el siguiente formato:

```
87463                    // Numero a
factorizar
295                      // Raíz de 'n'
6                        // Cota B
264                      // Límite M
[2 3 13 17 19 29 -1]     // Base de Factores

xi: -254
F(xi): 1
Parejas (basePrimo, exponente):
(2, 1)
(3, 1)
(17, 1)
(29, 2)
(-1, 1)
…
```

3. *Construcción y reducción de la matriz*

Construye la matriz con los vectores de exponentes de los resultados en el cuerpo $GF(2)$. Después procede a reducir la matriz con una variante de la reducción Gaussiana, aprovechando las características que ofrece GMP en cuanto a optimización de las operaciones y tratamiento de matrices reduciendo su complejidad.

4. *Solución*

Finalmente, haciendo uso del Algoritmo Extendido de Euclides buscamos factores no triviales del conjunto de soluciones obtenido en el apartado anterior. Cuantas más soluciones en el proceso de criba encontremos, más opciones tendremos en este apartado de encontrar un factor no trivial del número. Como buscamos, como mínimo, diez números más de los que necesitamos, la probabilidad de encontrarlo es muy alta.

Finalmente los resultados son impresos tanto por consola como en un archivo, con el nombre de 'factores.txt'.

El módulo de álgebra lineal ha sido desarrollado por Alberto Guillén y Alberto Megía para la asignatura de Criptografía y Teoría de Códigos, Facultad de Informática, Universidad Complutense de Madrid.

# Apéndice II: Archivos de configuración del CodeSwarm

**# SAMPLE.CONFIG**

```
# This is a sample configuration file for code_swarm

# Frame width
Width=640

# Frame height
Height=480
```

**# Las variables "Width" y "Height" se refieren al alto y ancho de la ventana que se abre cuando iniciamos el ejecutable del programa, en la cual se mostrará la representación gráfica del archivo log. Éstas serán también las dimensiones de los archivos de imagen en formato .png si aceptamos almacenar los "frames" de la secuencia, indispensable para poder generar un video de la misma (ver más adelante).**

```
# Input file
InputFile=data/sample-repevents.xml
```

**# La ruta del archivo log en formato XML de donde se obtiene el historial de eventos. Por defecto, el programa trae el archivo "sample-repevents.xml" que muestra un pequeño ejemplo de las capacidades de este programa.**

```
# Particle sprite file
ParticleSpriteFile=src/particle.png
```

**# Imagen en formato .png para definir los denominados anteriormente "puntos luminosos", que representan un archivo. Por defecto está asignada la imagen "particle.png", que simula una estrella, pero se puede cambiar por cualquier otra. Es recomendable que sus dimensiones sean las mismas (30x30 pixels).**

```
#Font Settings
Font=SansSerif
FontSize=10
BoldFontSize=14
```

**# Formato empleado para el texto que aparece en la ventana: leyenda, fecha y cadenas de caracteres que representan a las máquinas que intervienen en las transferencias de archivos.**

```
# Project time per frame
MillisecondsPerFrame=21600000
```

**# Intervalo de tiempo de proyecto por "frame" en milisegundos. Cada vez que transcurre el tiempo definido en este parámetro durante la ejecución del módulo Engine, le corresponde al programa hacer una nueva captura. El valor del ejemplo corresponde a 6 horas, ya que originalmente el programa está enfocado a revisar historiales de repositorios. En nuestro caso serán valores de segundos o minutos, en función del tiempo total de ejecución.**

```
# Maximum number of Background processes
MaxThreads=4
```

**# Número máximo de procesos en segundo plano. Este parámetro sirve para limitar los recursos que el programa empleará cuando se esté ejecutando.**

```
# Optional Method instead of MillisecondsPerFrame
#FramesPerDay=4
```

**# En lugar de utilizar la variable "MillisecondsPerFrame", se puede optar por emplear esta variable, que determina el número de "frames" capturados por cada día de ejecución del módulo Engine. Claramente enfocada para historiales de repositorios, puede servir para tareas extremadamente grandes.**

```
# Background in R,G,B
Background=0,0,0
```

**# Color del fondo de la ventana en formato RGB. Es negro por defecto.**

```
# Color assignment rules
# Keep in order, do not skip numbers. Numbers start
# at 1.
#
# Pattern:  "Label", "regex", R,G,B, R,G,B
# Label is optional.  If it is omitted, the regex
# will be used.
#
ColorAssign1="Docs",".*doc.*", 0,0,255, 0,0,255
ColorAssign2="Code1",".*src1.*", 0,255,255, 0,255,255
ColorAssign3="Code2",".*src2.*", 102,0,255, 102,0,255
ColorAssign4="Code3",".*src3.*", 255,0,0, 255,0,0
ColorAssign5="Code4",".*src4.*", 255,255,0, 255,255,0
ColorAssign6="Code5",".*src5.*", 119,68,119, 119,68,119
ColorAssign7="Code6",".*src6.*", 136,51,17, 136,51,17
ColorAssign8="Code7",".*src7.*", 250,110,110, 250,110,130
ColorAssign9="Code8",".*src8.*", 238,102,68, 238,102,68
ColorAssign10=".*src9.*", 238,68,119, 238,68,119
```

**# Asignación de colores en formato RGB a los diferentes tipos de los archivos transferidos en la ejecución del módulo Engine. Cada punto luminoso tendrá un color determinado dependiendo de estas sentencias. También se puede asignar el texto correspondiente a cada tipo/color de archivo en la leyenda que aparecerá situada en la esquina superior izquierda de la ventana.**

```
# Save each frame to an image?
TakeSnapshots=false

# Where to save each frame
SnapshotLocation=frames/code_swarm-#####.png
```

**# La variable booleana "Takesnapshots" indica si se guardará o no cada frame que se genere durante la ejecución de Codeswarm en formato de imagen .png. "SnapshotsLocation" designa la ruta en la que se guardarán estos archivos.**

```
# Draw names (combinatory) :
# Draw sharp names?
DrawNamesSharp=true
# And draw a glow around names? (Runs slower)
DrawNamesHalos=false

# Draw files (combinatory) :
# Draw sharp files
DrawFilesSharp=false
# Draw fuzzy files
```

```
DrawFilesFuzzy=true
# Draw jelly files
DrawFilesJelly=false
```

**# Estas variables definen la manera en la que se visualizarán los nombres correspondientes a cada máquina que interviene en la ejecución del módulo Engine y los puntos luminosos que representan a los archivos. Cada nombre puede aparecer nítido y/o brillante (ésta última propiedad afecta al rendimiento de la aplicación) y cada punto puede ser nítido, difuminado y/o con aparencia "gelatinosa".**

```
# Show the Legend at start
ShowLegend=true

# Show the History at start
ShowHistory=true

# Show the Date at start
ShowDate=true

# Show edges between authors and files, mostly for debug purpose
ShowEdges=false
```

**# Estas cuatro variables son parámetros booleanos que afectan a la información extra proporcionada en la ventana. Se puede definir si se desea que se muestren o no la leyenda, el histograma de volumen de archivos por intervalo en la parte inferior, la fecha (en la esquina inferior derecha) y líneas o aristas entre los nombres de las máquinas y los archivos, con propósitos de revisión y también puede afectar al rendimiento y la claridad de la secuencia.**

```
# Turn on Debug counts.
ShowDebug=false
```

**# Si se activa a true, cuenta el número de fallos o "bugs" que puede haber durante la ejecución de Codeswarm.**

```
# Natural distance of files to people
EdgeLength=25
```

**# Distancia en píxeles que separa los nombres de cada máquina a los archivos situados en su "órbita".**

```
# Amount of life to decrement
EdgeDecrement=-2
FileDecrement=-2
PersonDecrement=-1
```

**# Cada arista, archivo o nombre de máquina tienen un valor de vida que disminuye en cada nuevo frame (ver más abajo). Estos tres valores indican la rapidez a la que va desapareciendo cada elemento.**

```
#Speeds.
#Optional: NodeSpeed=7.0, If used, FileSpeed and PersonSpeed need not
#be set.
FileSpeed=7.0
PersonSpeed=2.0
```

**# Velocidad de desplazamiento de cada nodo (nombre de máquina o punto luminoso). Se puede optar por una variable que defina la velocidad de todos los nodos o dar un valor por separado.**

```
#Masses
FileMass=1.0
PersonMass=10.0
```

**# Masa correspondiente a cada nombre de máquinas y puntos. Éste**

**valor influye en el motor físico implementado para este programa, en concreto en la fuerza de atracción que ejerce cada elemento sobre los demás.**

```
# Life of an Edge
EdgeLife=250
# Life of a File
FileLife=200
# Life of a Person
PersonLife=255
```

**# Vida o tiempo que perduran los elementos (aristas, puntos y nombres de máquina) tras ser referenciados en el archivo log XML. Cuanto mayor es su valor, mayor es el tiempo que permanecen visibles a no ser que vuelvan a intervenir en alguna transferencia en cuyo caso su valor vuelve a ser el máximo. Si no son referenciados, desaparecen cuando su valor llega a cero.**

```
# Highlight percent.
# This is the amount of time that the person or
# file will be highlighted.
HighlightPct=5
```

**# Porcentaje de vida durante el cual el elemento (nombre de máquina o punto) aparece destacado, es decir, más luminoso.**

```
## Physics engine selection and configuration
# Directory physics engine config files reside in.
PhysicsEngineConfigDir=physics_engine
# Force calculation algorithms ("PhysicsEngineLegacy",
"PhysicsEngineSimple"...) :
PhysicsEngineSelection=PhysicsEngineLegacy
```

**# Selección del motor físico. Todos se encuentran en el directorio "physics_engine". Dependiendo del que se seleccione, las interacciones entre los elementos que componen la secuencia visual varían.**

```
# OpenGL is experimental. Use at your own risk.
UseOpenGL=false
```

**# Activación de la librería gráfica OpenGL para un mayor rendimiento. Su integración se encuentra en fase experimental en esta versión del programa, y debido a ello la gran mayoría de usuarios experimentan problemas graves que hacen imposible la ejecución de Codeswarm, por lo que se recomienda no activarla.**

Archivo de configuración Codeswarm utilizado en el proyecto:

**# SAMPLE.CONFIG**

```
# This is a sample configuration file for code_swarm

# Frame width
Width=1440

# Frame height
Height=900

# Input file
InputFile=data/activity49.xml

# Particle sprite file
ParticleSpriteFile=src/particle.png
```

```
#Font Settings
Font=SansSerif
FontSize=12
BoldFontSize=16

# Project time per frame
MillisecondsPerFrame=1600

# Maximum number of Background processes
MaxThreads=25

# Background in R,G,B
Background=0,10,30

# Color assignment rules
# Keep in order, do not skip numbers. Numbers start
# at 1.
#
# Pattern:  "Label", "regex", R,G,B, R,G,B
# Label is optional.  If it is omitted, the regex
# will be used.
#
ColorAssign1="Archivo .txt",".*txt.*", 255,255,255, 255,255,255
ColorAssign2="Script Perl",".*pl.*", 0,255,255, 0,255,255
ColorAssign3="Clave Publica",".*pub.*", 102,0,255, 102,0,255
ColorAssign4="Log XML",".*xml.*", 255,0,0, 255,0,0
ColorAssign5="Ejecutable/Sin extension",".*.*", 255,255,0, 255,255,0

# Save each frame to an image?
TakeSnapshots=true

# Where to save each frame
SnapshotLocation=/media/IOMEGA_HDD/frames/code_swarm-#####.png

# Draw names (combinatory) :
# Draw sharp names?
DrawNamesSharp=true
# And draw a glow around names? (Runs slower)
DrawNamesHalos=false

# Draw files (combinatory) :
# Draw sharp files
DrawFilesSharp=true
# Draw fuzzy files
DrawFilesFuzzy=true
# Draw jelly files
DrawFilesJelly=false

# Show the Legend at start
ShowLegend=true

# Show the History at start
ShowHistory=true

# Show the Date at start
ShowDate=false

# Show edges between authors and files, mostly for debug purpose
ShowEdges=false

# Turn on Debug counts.
```

```
ShowDebug=false

# Natural distance of files to people
EdgeLength=50

# Amount of life to decrement
EdgeDecrement=-2
FileDecrement=-1
PersonDecrement=-1

# Speeds.
FileSpeed=25.0
PersonSpeed=0.0

# Masses
FileMass=4.0
PersonMass=1000.0

# Life of an Edge
EdgeLife=250

# Life of a File
FileLife=700

# Life of a Person
PersonLife=25000

# Highlight percent.
# This is the amount of time that the person or
# file will be highlighted.
HighlightPct=10

## Physics engine selection and configuration
# Directory physics engine config files reside in.
PhysicsEngineConfigDir=physics_engine
# Force calculation algorithms ("PhysicsEngineLegacy",
"PhysicsEngineSimple"...) :
PhysicsEngineSelection=PhysicsEngineLegacy

# OpenGL is experimental. Use at your own risk.
UseOpenGL=false
```

# Apéndice III: Artículo Jornadas de Paralelismo

El siguiente artículo ha sido aceptado para las XXII Jornadas de Paralelismo celebradas en la Universidad de la Laguna, Tenerife, España, los días 7, 8 y 9 de Septiembre de 2011.

Se incluye en el apéndice de la memoria para uso estrictamente personal.