Universidad Complutense de Madrid

Facultad de Informática U.C.M

Planificación y Optimización Paralela de Redes Logísticas de Transporte

Proyecto de Sistemas Informáticos

Realizado por: Adrián García Romero, Miguel Mena Jiménez y Alejandro Soto Rebollo

Dirigido por: José Jaime Ruz Ortiz

Facultad de Informática

Facultad de Informática

Prefacio

En este trabajo se detalla el proceso de desarrollo de una aplicación que permite realizar análisis de sensibilidad de una red logística de transporte de gas de forma paralela en varias máquinas y a través de la red. Esto incluye tanto la creación de una aplicación de escritorio que permita enviar los datos con los que se quiere realizar el análisis como un servicio web que ejecuta el resolutor IBM ILOG CPLEX y que realiza efectivamente el cálculo. En las siguientes secciones se procederá a explicar primero que es y como funciona y se modela una de estas redes de gas. Seguidamente pasaremos a comentar como usamos CPLEX y cual es el método para solucionar los modelos. A continuación se hará una breve introducción a ciertos conceptos que se necesitan conocer para poder comprender como funciona el paralelismo de la aplicación. Finalmente entraremos a fondo en la implementación de la aplicación para acabar con las conclusiones que hemos extraído, incluyendo posibles ampliaciones del proyecto.

Palabras clave: Optimización, CPLEX, Paralelismo, Servicios Web, Modelado

In this work we detail the developing process of program which allows sensitivity analysis of a gas distribution network. These analysis are made in parallel on multiple computers and across the net. This incudes the creation of a desktop application that can send the data you want to perform the analysys with and a web sevice that runs IBM ILOG CPLEX solver and that actually performs the calcullation. The following sections will explain what is and how one of this gas networks works and is modeled. After that, we will talk about how we use CPLEX and the algorith it uses to solve the models. Next, we will briefly introduce certain concepts needed to understand how the application exploits paralelism. Finnally we will enter fully in the implementation, ending with the conclusions we have obtained, including possible extensions of the project.

Keywords: Optimization, CPLEX, Paralelism, Web Services, Modelling

Propuesta

Introducción

Una de las primeras tareas para las que se diseñaron los ordenadores fue la resolución de problemas matemáticos, ya sea por la excesiva complejidad de los mismos o por la cantidad de tiempo y recursos que consumiría realizarlos manualmente. Con el aumento de la potencia de cálculo de los procesadores, el uso que se ha podido dar de los mismos ha llegado a abarcar casi todos los ámbitos de la vida. Uno de los más importantes es el modelado de sistemas. Con la ayuda de modelos matemáticos bien diseñados se pueden predecir comportamientos de lo más variado. Desde fenómenos aparentemente caóticos cómo podrían ser los relativos a la meteorología hasta aplicaciones más concretas cómo los planes estratégicos de las empresas. La necesidad de conocer el comportamiento de un sistema es de capital importancia y puede decidir la dirección de inversiones clave.

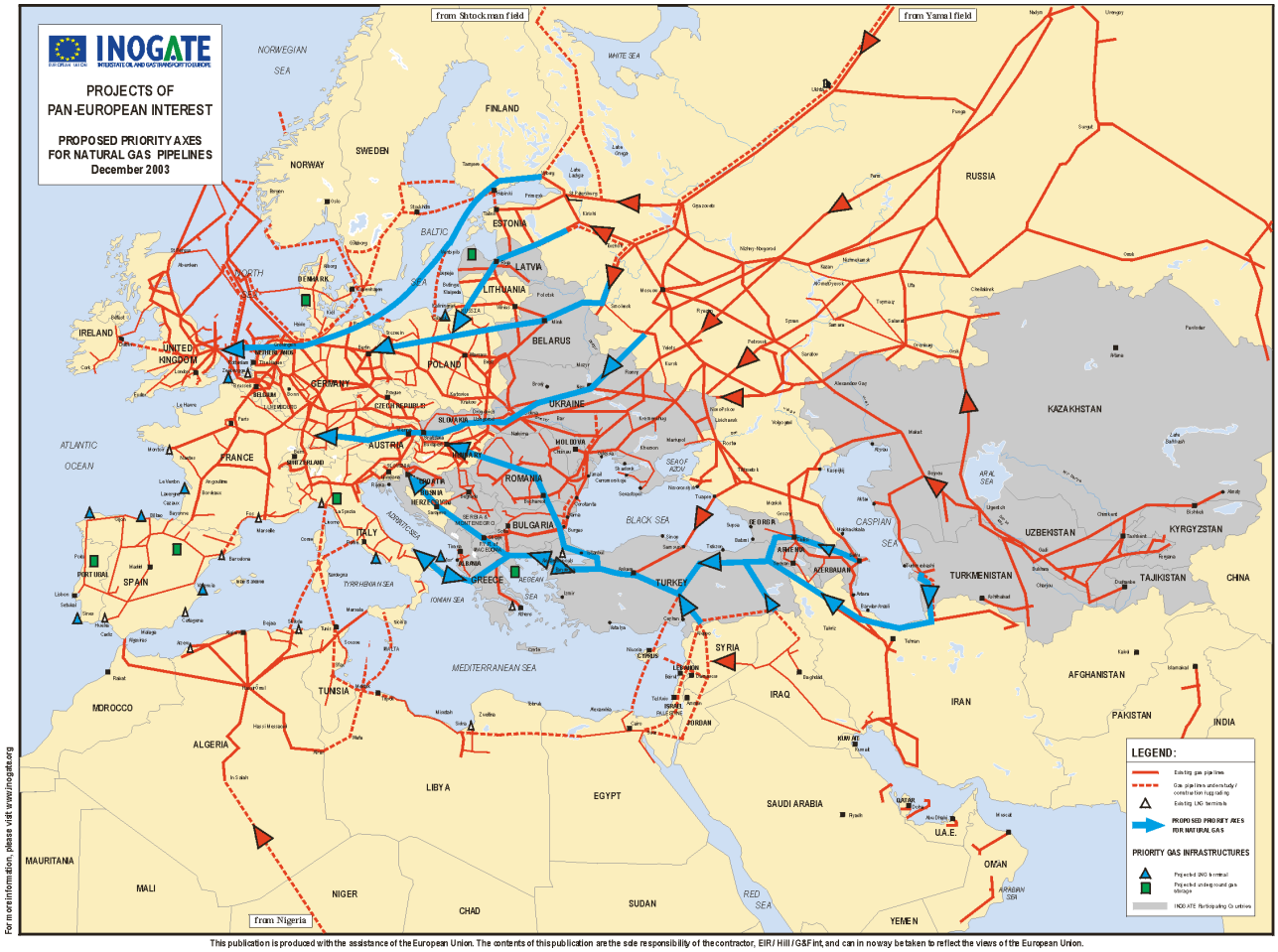
Objetivo y alcance

En particular, la finalidad de este proyecto es la de poder realizar el análisis de sensibilidad de una planificación óptima obtenida para un sistema logístico de distribución de gas natural.

Un análisis de sensibilidad consiste en comprobar cómo se comporta la solución de un modelo si varían determinados parámetros que afectan al mismo. De esta forma se pueden extraer conclusiones sobre qué parámetros son más críticos para el sistema, como se comportaría en situaciones que no son las ideales o, simplemente, cuales hacen que funcione con el mínimo costo posible. Sin embargo, el factor más importante para este proyecto que permiten estos análisis es conocer la estabilidad del sistema, esto es si a pesar de obtenerse una solución muy buena en términos de coste, esta empeora sensiblemente en cuanto se varíen mínimamente los parámetros.

Una red de distribución logística de gas es una clase de red de abastecimiento en la que se planifica el flujo de gas. Consta de una red principal que transporta el gas a alta presión hasta puntos clave de la geografía de un país y una red secundaria de baja presión que lo distribuye desde estos últimos hasta las industrias, los hogares, etc. El suministro de gas de un país es un sector estratégico e históricamente se ha realizado siguiendo un modelo centralizado en el que una única empresa controla todos los aspectos del mismo. En los últimos años, sin embargo con la liberalización del

mercado del gas natural, han surgido nuevas empresas de distinto tipo (proveedores, distribuidores...) que compiten entre sí y mantienen los precios del gas a valores de mercado. En cualquier caso, por encima de estas empresas existe un ente conocido como gestor técnico del sistema que es quien regula la operación de los demás agentes gasistas en las redes primaria y secundaria, así como en las plantas de regasificación y almacenes subterráneos. Este gestor se encarga de asegurar que ningún punto de la red quede desabastecido. Por tanto esta clase de redes está guiada por demanda.

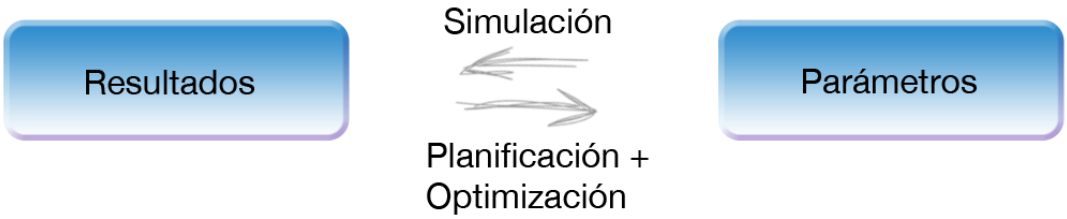


Red principal de distribución de gas en Europa, Oriente Medio y el Norte de África

Planificar una de estas redes significa precisamente gestionar el tráfico de gas de forma que, satisfaciendo la demanda, se mueva la menor cantidad de gas por el sistema. Esto a su vez implica que las estaciones de compresión que realizan esta tarea estarán trabajando el mínimo tiempo necesario. Teniendo en cuenta el costo de operación de las mismas esto puede significar un enorme beneficio para la empresa.

El modelo matemático a utilizar para realizar una planificación óptima de una red difiere del de la simulación de esta. Un modelo de planificación busca satisfacer una demanda minimizando el coste de su distribución, mientras que un modelo de simulación se preocupa de la viabilidad física para unas condiciones dadas de la red. La primera se basa en el uso de técnicas de satisfacción de restricciones y optimización matemática, mientras que la segunda usa técnicas de integración numérica de las ecuaciones diferenciales que rigen el comportamiento del gas a lo largo de los dispares elementos de la red.

Por tanto, una planificación es un cálculo a futuro de como hay que operar un sistema para que se satisfaga una demanda conocida. Lógicamente, las soluciones óptimas de planificación deben ser posibles a nivel físico, por lo que es posible asegurar la solución propuesta por el optimizador mediante la simulación.



Debido a que el sistema no es totalmente lineal, la sensibilidad no puede resolverse tan solo mediante técnicas analíticas de los sistemas de optimización lineal y debe realizarse análisis empírico de sensibilidad usando técnicas de barrido para los parámetros de interés. Este análisis consiste en ejecutar el mismo modelo con diferentes valores del citado parámetro.

Cada una de las ejecuciones con valores diferentes del parámetro son independientes entre sí, por tanto existe la posibilidad de explotar un alto grado de paralelismo. Aprovechando esto y usando el paradigma de programación distribuida, este proyecto permite ejecutar cada instancia del problema en una máquina diferente, de tal forma que el tiempo necesario para resolver el análisis de

sensibilidad se reduce de manera importante. Si suponemos que el problema consta de los parámetros x_o a x_s, se envía a cada máquina distintas instancias de la misma lista a excepción del parámetro sobre el que deseamos hacer el barrido, llamémoslo x_i. Este cambia para cada uno de los envíos, que se realizan paralelamente, como se muestra en la siguiente figura:

Relación con los estudios

El proyecto tiene relación con varias asignaturas cursadas durante la carrera además de profundizar en otros campos que se tratan solo de manera tangencial en la misma:

Análisis Matemático y Ampliación de Calculo

La optimización de funciones (hallar sus puntos máximos y mínimos) en una de las primeras cosas que se estudian en las asignaturas más orientadas a la matemática pura durante los primeros cursos. Aunque la aplicación usa algoritmos mucho más refinados para encontrar máximos en espacios multidimensionales, el paradigma que usa para hacerlo es la final el mismo.

Programación Lógica

Como se explicará más adelante, el lenguaje que se usa para resolver el modelo es OPL. Este es un lenguaje de programación por restricciones, que no es más que un tipo de programación declarativa. Ya que el Prolog, que es el lenguaje que se usa en la asignatura de Programación Lógica también es un lenguaje declarativo, el método de trabajo que se usa con ambos es similar.

Redes

A la hora de tratar con la parte web de la aplicación, ha habido que tratar con diferentes protocolos de comunicación. Aunque algunos son totalmente nuevos y hemos tenido que realizar un estudio previo de su viabilidad para el proyecto, otros, como puede ser el caso de TCP/IP, se ven ampliamente en la asignatura de redes y esto ha permitido una más rápida implementación.

Estadística

El análisis y el tratamiento de datos es una tarea vital para este proyecto. Las lecciones aprendidas en esta asignatura sobre como realizar esta tarea han permitido una mayor comprensión de la envergadura del mismo.

Investigación Operativa

En esta asignatura se realiza una introducción a los procesos de planificación de redes logísticas y, entre otros algoritmos, el método Simplex. Todo esto esta enormemente relacionado con este proyecto como se verá más adelante. Cómo definir modelos de programación lineal es también una de los aspectos que se tratan y ha sido de gran ayuda a la hora de definir el modelo de una red de gas.

Capítulo 1:
Conceptos y Estado del Arte

Introducción al Capítulo

El objetivo de este capítulo es hacer una breve introducción a una serie de conceptos necesarios para comprender la función que realiza la aplicación. Los sistemas de suministro de gas o cual es el significado de su optimización no suelen ser conocidas por el público general. Además todos los apartados reflejados en el Capítulo 2: Aplicación, tienen su correspondencia directa con un apartado de este capítulo para facilitar la consulta.

Posteriormente hablaremos del estado del arte en lo que se refiere a optimización matemática y computación distribuida utilizando servicios web. Estos dos factores son definitorios en este proyecto y hemos creído oportuno comentar como funcionan. Mención especial merece el resolutor CPLEX sobre el que se ejecutan algoritmos para resolver problemas de optimización lineal entera mixta (MILP). Se ha optado por dejar de lado temas que no se tocan en el proyecto o que no sean de interés para el mismo. Aunque ciertamente se podría escribir cientos de líneas sobre que es un servicio web y cómo funcionan sus diferentes tipos, hemos creído oportuno mencionar únicamente el estado de las tecnologías que usamos y dar unas cuantas pinceladas de las que no utilizamos pero se llegaron a considerar para el proyecto. Esto último se realizará en el Capítulo 3: Desarrollo.

Gran parte de la carga de trabajo del proyecto ha consistido en investigar y comprender estos sistemas, por lo que esta sección sirve a su vez como una demostración de los conceptos asimilados.

Redes Logísticas de Transporte de Gas

Antes de entrar en más detalle en la aplicación, vamos a hablar de los objetos con los que trabaja la misma. Las redes logísticas de gas son una clase especial de redes logísticas en las que se planifica como transita el gas por una serie de conducciones desde los puntos de suministro (Ya sean plantas de regasificación donde llegan barcos cargados de gas natural licuado, como conexiones internacionales de gasoductos) hasta los puntos de demanda.

La finalidad de las mismas es satisfacer esta demanda. Queda fuera del alcance de este proyecto determinarla, pero merece la pena mencionar que es una tarea que se realiza por métodos estadísticos y en la que intervienen factores de lo más variado: Número de personas a satisfacer, industrias de la zona, época del año, tiempo meteorológico, etc, consiguiendo valores que se aproximan mucho a la demanda real. Hablamos por tanto de un modelo guiado por demanda. En la vida real, el suministro de gas es un sector estratégico para el país y por tanto incurrir en el desabastecimiento de la población o de la industria puede suponer fuertes multas para la empresa encargada de realizarlo. Es más, no solo el desabastecimiento: estas empresas están obligadas por ley a disponer de una cantidad mínima de reserva para tener margen de maniobra en el caso de que surgiese algún imprevisto. Conociendo todo lo anterior queda claro que cualquier modificación que se realice en el sistema tiene que estar precedida de suficientes estudios de viabilidad que la avalen.

Por otro lado, a veces no basta con que el modelo cumpla los requisitos para ciertos parámetros porque, como en todo, existe cierta incertidumbre en el proceso. Una tormenta puede retrasar la llegada a puerto de un barco, se tienen que tener en cuenta posibles paradas por mantenimiento que hagan que una parte del sistema deje de conducir gas… Por tanto el sistema tiene que ser tolerante a estos cambios y seguir pudiendo suministrar la capacidad requerida. Por decirlo de otra forma, aunque un sistema produzca una gran cantidad de beneficio para unos parámetros concretos, si es inestable, cualquier imprevisto provocará una multa que dará al traste con toda la planificación.

La inherente imprevisibilidad de estos acontecimientos hace imposible saber que efectos producirían en el suministro a no ser que se haga alguna clase de estudio previo. Un análisis de sensibilidad como el que propone este proyecto es idóneo para comprobarlo. Por poner un ejemplo, sigamos con el caso del barco que se retrasa. Si tenemos un modelo al que llega una cierta capacidad de suministro y que funciona durante n periodos. ¿Qué ocurre si es necesario que opere durante más tiempo con la misma capacidad? Basta con variar el parámetro que controla los periodos entre n y n+k y comprobar a partir de cuál de ellos el problema deja de tener solución. Esta sería la tolerancia del sistema al desabastecimiento.

Pasemos a hora a conocer en detalle que compone una red de suministro de gas. Cada una de estas redes cuenta con una serie de elementos básicos.

Componentes

Suministro

Como ya se ha dicho, los puntos de suministro son los lugares por los que entra gas al sistema. Esto puede tomar varias formas, desde gasoductos conectados con otros países, puertos a los que llegan buques cargados de gas licuado (previamente es necesario convertirlo de nuevo en gas en una planta regasificación) o directamente, puntos de extracción. Sin embargo y a la hora de planificar, el único parámetro que nos interesa es cuanta capacidad pueden suministrar.

Conducciones

Las conducciones o gasoductos son las tuberías por las que circula el gas. Pueden tener uno o más puntos de demanda en los que el gas abandona el sistema y que es necesario satisfacer de forma que no se produzca inanición en los consumidores. Además es necesario que la presión del gas en el interior de los mismos esté dentro de los parámetros de seguridad.

Almacenamiento

El gas se almacena de manera temporal en determinadas cantidades en unas instalaciones subterráneas. En estas instalaciones se puede introducir o extraer gas a una tasa variable y tienen una capacidad máxima que no se puede superar.

Estaciones de compresión

Mediante las estaciones de comprensión se aplica presión al gas de manera que puedan vencer las diferencias de nivel y el rozamiento de las tuberías. Si la necesidad de utilizar las estaciones de compresión se reduce, el consumo general es menor y por tanto mayor beneficio para la empresa siempre que se satisfaga la demanda. Por supuesto las estaciones de compresión pueden trabajar a varias intensidades y su consumo depende como es obvio de estas.

Modelo de la red

Crear un modelo matemático de la red es siempre una tarea complicada y puede crearse un modelo tan complejo como se desee.

Para empezar hay que comentar que el modelo se mueve en dos dimensiones: espacial y temporal. En la primera entra en juego la colocación de los diversos elementos dentro del modelo. La dimensión temporal viene determinada por el carácter multiperiodo del modelo. Para cada periodo habrá que definir unas variables de entrada para cada elemento y el modelo se encargará de devolver una salida en función de las mismas y posiblemente del estado en el periodo anterior. Vamos a analizar cada elemento en función de los parámetros citados, a saber variables de entrada, variables de salida y restricciones que debe cumplir. Este sistema de restricciones nos vendrá bien luego a la hora de definir le modelo en un lenguaje de optimización matemática.

Suministro

El modelo de suministro consta únicamente de una variable que índica la cantidad de gas que entra en el sistema en cada periodo y punto de suministro. La única restricción que se debe cumplir es que no exceda la capacidad límite del sistema.

Conducciones

En esta versión del modelo se contemplan conducciones en las que el gas puede circular en ambas direcciones, pero solo en una de ellas en un periodo dado. Por tanto:

- Variables de entrada: La cantidad de gas que hay en la tubería en el instante cero.
- Variables de salida: Por cada periodo, la dirección en la que circula el gas, el flujo de entrada y el de salida
- Restricciones:
  - El flujo de salida en un instante dado es igual al flujo de entrada en ese instante más la cantidad de gas que contenía la tubería en el instante anterior menos la cantidad que queda en el instante actual. Hay que restar además el gas que sale de la tubería hacia el exterior del sistema para satisfacer una demanda.
  - La cantidad de gas que hay en la tubería no excede las limitaciones estructurales de la misma

Almacenamiento

El modelo de almacenamiento también es bidireccional, esto permite tanto la introducción de gas en el depósito como su extracción, pero una vez más solo una de las dos operaciones está permitida por periodo.

- Variables de entrada: Cantidad de gas en el almacenamiento en el instante inicial.
- Variables de salida: Cantidad de gas, dirección y tasa de entrada/salida en cada periodo
- Restricciones:
  - El almacén no excede una capacidad máxima.
  - El estado del almacén en un instante es el estado en el instante anterior más el flujo de entrada/salida

Estaciones de compresión

El modelo permite que cambie la dirección de compresión de una estación en cada período, si es necesario. Ya que se trata del objeto cuyo consumo se intenta modelar, es el más complicado de todos.

- Variables de entrada: Número de posiciones de funcionamiento, flujo mínimo y máximo que soporta cada una de ellas y consumo de las mismas.
- Variables de salida: Dirección de bombeo y cantidad bombeada.
- Restricciones:
  - La cantidad bombeada en un momento dado está dentro de los parámetros de seguridad y se bombea en una única dirección.

Ejemplo de una estación de compresión. Nótese que la capacidad bombeada puede cambiar de sentido

Una vez definidos los modelos de los elementos individuales hay que modelar el sistema en sí. Al final, este no es más que un sistema de interconexiones entre cada uno de ellos. Se debe cumplir que en cualquiera de los nodos del sistema (Lugares en los que se conectan dos o más elementos) la cantidad de gas que entre y salga del mismo sea igual. Por lo tanto y para seguir con el método anterior se añade una restricción extra por nodo reflejando esta situación.

Programación Matemática

Programación lineal

La programación lineal es un tipo de programación matemática para resolver problemas lineales, en especial problemas logísticos. La programación lineal estudia la optimización (minimización o maximización) de una función lineal que satisface un conjunto de restricciones lineales de igualdad y/o desigualdad.

Definimos un problema lineal como un problema que puede ser representado de la forma canónica:

Minimizar:	$c^T x$
Sujeto a:	$Ax \leq b$
y:	$x \geq 0$

donde x es el vector de variables a ser determinadas, c^T y b vectores de coeficientes conocidos (c^T traspuesto) y A una matriz de coeficientes conocidos a su vez. Esta clase de problemas pueden ser resueltos en tiempo polinomial.

Un modelo de programación lineal consta de varias partes:

- Variables de decisión: Representan los elementos del sistema a modelar que son controlables por el decisor.
- Restricciones: Representan las limitaciones de los recursos o las imposiciones físicas de la realidad. Se representan como ecuaciones o inecuaciones. Para un modelo de gas, podrían ser las capacidades máximas de los almacenes o el límite de presión de las estaciones de compresión.
- La función objetivo: Es el valor que se intenta maximizar o minimizar. En nuestro caso es la función que define el consumo de las estaciones de compresión.

Programación lineal entera mixta (MILP)

Si en un problema de de programación lineal se exige que algunas de las variables desconocidas (el vector x en el ejemplo anterior) sean enteras, nos encontramos con un problema de programación lineal entera mixta. Esto presenta dos inconvenientes. Por un lado esta clase de problemas son NP-difíciles. Por el otro un análisis de sensibilidad sobre sus parámetros no puede hacerse analíticamente. El primero de los inconvenientes sin embargo no es tal para este proyecto en particular debido a que existen resolutores comerciales de problemas MILP muy eficientes como CPLEX. Para llevar a cabo esta tarea dispone de métodos como el de *branch & bound* y el de los planos cortantes que explicaremos en la siguiente sección.

Una vez definidas estas partes, el problema y su solución puede representarse en un espacio de n dimensiones, siendo n el número de variables de decisión. Las restricciones acotan una parte de este espacio y dan lugar a una "región de factibilidad". Si esta es vacía entonces el problema no tendrá solución. Si no lo es, dependiendo de si está acotada podemos hablar de problemas acotados (con una o múltiples soluciones) y no acotados (con infinitas soluciones).

Programación por restricciones

La programación por restricciones es un paradigma de programación en el cual las relaciones entre variables vienen dadas por restricciones. La principal diferencia entre este paradigma y el más clásico de programación imperativa es que las restricciones no se ejecutan secuencialmente como es el caso de las instrucciones en un programa imperativo. Esto hace que la programación por restricciones sea un tipo de programación declarativa.

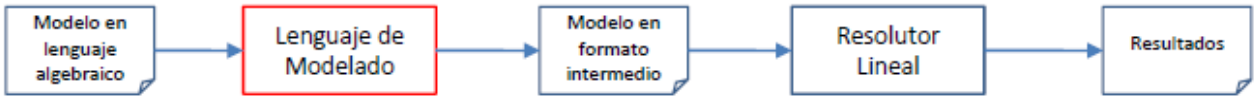
Como se ha explicado antes, el modelo del sistema de gas se expresa como un conjunto de variables y otro de restricciones que simulan el comportamiento. Usando el modelo de programación lineal descrito en el apartado anterior es posible formular todo el modelo como un programa, de tal forma que pueda ser analizado por un intérprete y resuelto. Sin embargo esto puede no ser trivial como veremos a continuación.

Hacia OPL

La programación lineal y la optimización combinatoria son problemáticas generales y comunes en muchas áreas de la ingeniería y la computación, como la planificación, secuenciación, configuración, diseño, administración y asignación de recursos. Sin embargo, es un reto el hecho de expresarlos, formularlos y formalizarlos adecuadamente.

Lenguajes algebraicos de modelado

Los lenguajes de modelado de problemas de optimización son lenguajes declarativos con una sintaxis próxima a la especificación matemática de estos de problemas. En esencia constituyen una interfaz de programación declarativa a través de la cual se utiliza el resolutor lineal de una forma más cómoda y productiva para el diseñador. Suelen disponer de recursos específicos para ayudar a la depuración del modelo. En el siguiente esquema hemos representado su ubicación en el proceso de resolución de un problema de optimización:



Aunque existen muchos lenguajes algebraicos de modelado para problemas de programación lineal (tales como Lingo, GAMS, o OML de Microsoft), se ha optado por utilizar para este proyecto el lenguaje OPL (de las siglas en inglés de "*Optimization Programming Language*"), un lenguaje con una gran capacidad expresiva cuyo principal interés reside en el soporte para programar con restricciones y planificación. Además, cuenta con interesantes posibilidades de interconexión e integración con fuentes de datos y entornos de desarrollo de software. Así pues, es la opción ideal para administrar aplicaciones de localización, distribución y asignación de recursos, como es el caso de la tarea aquí desarrollada.

OPL

OPL es un lenguaje de modelado diseñado específicamente para problemas de optimización. La principal diferencia con otros lenguajes de modelado presentes en el mercado es que incluye soporte tanto para el propio modelado y la búsqueda de la solución como para la programación de restricciones. Por decirlo de otra manera, OPL permite especificar no solo el problema sino también el procedimiento de búsqueda asociado a ese problema. Por tanto OPL aúna tanto la programación matemática como la programación por restricciones de la que hemos hablado antes.

Las limitaciones e inconvenientes derivados del uso de OPL (tales como restricciones SOS1 y SOS2 pueden ser superadas manipulando el modelo generado con OPL desde un lenguaje (C#, Java, C++) que disponga de una interfaz (API) con OPL.

CPLEX

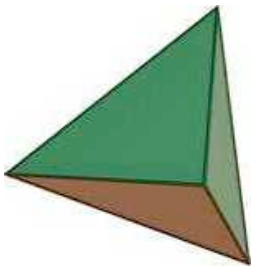
IBM ILOG CPLEX es un software de optimización creado por Robert E. Bixby y que persigue resolver problemas de programación lineal usando diferentes lenguajes para describirlos, entre ellos OPL, y utilizando el método simplex para hallar una solución. Este es un algoritmo relativamente complejo que vamos a explicaremos más adelante. Sobre CPLEX merece la pena mencionar que hoy en día es uno de los optimizadores más utilizados tanto a nivel de investigación como en el mundo empresarial. Debido a la potencia del mismo y a la gran cantidad de posibilidades que ofrece no es una herramienta al alcance del público general debido a que comprar una licencia para el uso completo del mismo requiere una importante inversión. La posibilidad de poder trabajar con CPLEX es uno de los aspectos más llamativos del proyecto y gran parte del tiempo y del esfuerzo invertido en el mismo ha recaído en comprender como funciona el optimizador y cuales son las posibilidades que ofrece.

La versión puesta a nuestra disposición incluye una interfaz basada en Eclipse que permite facilitar el uso, pero debido a la naturaleza de la aplicación que hemos desarrollado, se ha tenido que prescindir de la misma y utilizar CPLEX como una biblioteca que se añade a Visual Studio y que permite realizar las llamadas desde C#. Aunque realizar ciertas tareas requiere un conocimiento más profundo de CPLEX, la flexibilidad que esto permite compensa de sobra este inconveniente.

Un problema de programación lineal puede, para ser resuelto con CPLEX, dividirse en dos partes: Un fichero con el modelo y la extensión .mod y otro con los datos que se van a ejecutar en dicho modelo y con la extensión .dat. Aunque esta división no es absolutamente necesaria, facilita la tarea a la hora de hacer trabajar un mismo modelo con diferentes baterías de datos como se da en el caso de este proyecto.

Algoritmos: El método simplex

Para explicar como funciona este algoritmo, volvamos por un momento a la representación gráfica de un modelo de programación lineal. La región del espacio definida por las restricciones no es más que un objeto geométrico de n dimensiones limitado por una serie de "caras". Esta estructura se llama Politopo y es la generalización a n dimensiones de los polígonos (dos dimensiones) y los poliedros (3 dimensiones).



Un tetraedro es un 3-politopo, un espacio tridimensional limitado por 4 planos. En un modelo de programación lineal cada cara representaría una restricción

En especial, para el método simplex nos interesan los politopos convexos. En un politopo convexo dos puntos cualesquiera del mismo quedan unidos por una línea que está completamente contenida dentro del propio politopo. Los límites exteriores del politopo son denominados hiperplanos, que una vez más no son más que la generalización del plano a n dimensiones (De hecho un politopo de n dimensiones está limitado por hiperplanos de n-1 dimensiones) Un hiperplano divide el espacio de dimensión n en dos hemiespacios de la misma dimensión que el espacio original. Se cumple además que estos hemiespacios son convexos.

Con todo lo anterior podemos llegar a la representación de un politopo convexo de n dimensiones como una intersección de hemiespacios. Esta intersección puede representarse de forma matricial, ya que un hemiespacio se representa con una inecuación del tipo

$$a_1x_1+a_2x_2+...+a_nx_n \leq b$$

y un politopo como una colección

$$a_{11}x_{11}+a_{12}x_{12}+...+a_{1n}x_{1n} \leq b_1$$
$$a_{21}x_{21}+a_{22}x_{22}+...+a_{2n}x_{2n} \leq b_2$$
$$a_{31}x_{31}+a_{32}x_{32}+...+a_{3n}x_{3n} \leq b_3$$
$$.$$
$$.$$
$$a_{n1}x_{n1}+a_{n2}x_{n2}+...+a_{nn}x_{nn} \leq b_4$$

Siendo los coeficientes a números reales. Al final se puede reducir cualquier problema de programación lineal a este esquema de minimizar

$$c \cdot x$$

Sujeto a:

$$Ax = b \text{ con } x_i \geq 0$$

siendo x el vector de dimensión n con las variables de decisión del programa, c otro vector de dimensión n con los coeficientes de la función objetivo, A una matriz de dimensión n·p y b un vector de dimensión p con constantes. A esta representación se le llama forma estándar. Debido a que A multiplicado por x es igual a b, se hace necesario introducir una columna más en la matriz del problema original con variables que permitan expresar la igualdad. Estas variables se denominan variables de holgura si suman (para convertir ≤ en =) o de exceso si restan (para convertir ≥ en =) He aquí un ejemplo:

$3x_1+4x_2 \leq 6$ pasa a ser $3x_1+4x_2+x_3 = 6$ (x_3 variable de holgura)

$3x_1+4x_2 \geq 6$ pasa a ser $3x_1+4x_2-x_3 = 6$ (x_3 variable de exceso)

Volviendo a la representación espacial, se puede demostrar por métodos matemáticos que si la función objetivo presenta un mínimo dentro de la región factible (dentro del politopo), este valor se presenta también en al menos uno de sus puntos extremos, esto es en uno de los vértices del politopo. Por tanto, encontrar el valor que hace que la función objetivo devuelva un valor óptimo podría limitarse a explorar todos los vértices del politopo. Por desgracia, el número de vértices crece de forma exponencial al número de restricciones y debido a esto resulta impracticable utilizar este método en cualquier problema de programación lineal medianamente complejo.

Sin embargo y asimismo, también se puede demostrar que si uno de estos vértices no es el mínimo de la función objetivo existe al menos una arista que sale del vértice a lo largo de la cual la función es estrictamente decreciente. Sabiendo esto, el algoritmo simplex explora el politopo moviéndose entre vértices cada vez con un mejor valor de la función objetivo. Se procede de este modo hasta que se encuentra un mínimo o se da con un vértice no acotado, en cuyo caso el problema no tiene solución. Debido a que el número de vértices es finito el algoritmo siempre termina, aunque esto puede llevar ciertamente bastante tiempo si hubiese que visitarlos todos. En cualquier caso, como la dirección de exploración es siempre la misma se espera que el número de vértices visitados sea pequeño.

Para hallar la solución del programa lineal hace falta por tanto proceder en dos fases: Primero encontrar un vértice inicial desde el que empezar a buscar y segundo aplicar el algoritmo que se mueve entre los vértices. La primera parte puede o no ser trivial y en caso de no serlo se resuelve aplicando una versión modificada del algoritmo al problema inicial.

Eficiencia

La eficiencia del algoritmo en el caso medio está en el orden polinómico.

Algoritmos: El método *Branch & Bound*

El método simplex funciona bien a al hora de resolver problemas de programación lineal. Sin embargo, como ya se ha comentado, los problemas a los que nos enfrentamos en este proyecto requieren que ciertas variables sean enteras, lo que los convierte en problemas enteros lineales mixtos. En este escenario, el método simplex no es suficiente para hallar una solución, sin embargo puede utilizarse junto con otros algoritmos para hallar una solución de manera eficiente: Uno de estos algoritmos es el de *branch & bound.*

La idea de este método consiste en dividir el problema MILP en varios grupos. A continuación, estimando una serie de cotas para los mismos, se pueden descartar algunos de estos grupos. De esta forma se descartan candidatos en masa hasta que únicamente queda un conjunto con una posible solución o no quedan más conjuntos, en cuyo caso el problema sería irresoluble. Las fases de división (*Branch*) y acotación (*Bound*) son las que dan nombre al algoritmo.

Para un problema MILP, este es el esquema que permite encontrar la solución utilizando el método de *branch & bound*:

- Partiendo del problema inicial, se relajan las restricciones, de tal manera que se permite que las variables enteras tomen valores reales. Se haya la solución de este problema de programación lineal (En nuestro caso, usando el método Simplex) obteniendo una valor óptimo (cota inferior del óptimo para el problema MILP, ya que este solo añade restricciones extra al problema de programación lineal) y los valores de los parámetros (reales) que permiten alcanzar esa solución.
- Utilizando estos valores reales, se divide el problema en dos o más subproblemas. Por ejemplo, si en el problema inicial relajado, x debía tomar un valor mayor o igual que 0 y para el óptimo toma un valor de 21.2, podemos dividir el problema inicial relajado en dos: uno en el que x está en el rango de 0 a 21 y otro en el que es mayor que 21.
- Evaluamos los subproblemas con el método simplex una vez más y continuamos bifurcando para la misma u otras variables hasta que no se satisfaga que solo exista un valor entero impuesto por las restricciones de la bifurcación. Cuando lleguemos a este punto tenemos una solución entera candidata a resolver el problema.
- La bifurcación continúa por las distintas ramas del árbol. Sin embargo, si ya tenemos un candidato, la evaluación de cualquier nodo que de un valor peor que la solución candidata implica que podemos descartar esa rama con seguridad debido a que no contiene ninguna solución mejor que la que tenemos, incluso si relajásemos las restricciones.
- Si llegamos a otra solución entera mejor que la candidata, esta se actualiza y se sigue el proceso. Así mismo, si un nodo no tiene solución se descarta.
- Llegaremos a un punto en el que no existan nodos por explorar. La solución candidata en este momento representa el óptimo del problema MILP.

Algoritmos: El método de planos cortantes

Otro método que se usa en conjunción con el anterior para resolver problemas MILP. En este caso, se comienza igualmente con una relajación de las restricciones del problema. Con el problema ya relajado, se usa un resolutor de problemas de programación lineal. Si está solución asigna valores enteros a las variables, es también la solución óptima del problema sin relajaciones MILP. En caso contrario, se añade una restricción adicional al mismo. Si, volviendo a la representación gráfica del problema, vemos el espacio de soluciones como un politopo, esta nueva restricción no será más que un plano que la cortará y lo restringirá aún más. El método se repite de nuevo sobre el nuevo politopo.

Hacen falta métodos específicos para cada clase de problema si queremos encontrar planos que formen caras del politopo de soluciones enteras, ya que estos son los que más restringen el espacio de soluciones. Se ha demostrado que siempre existe un plano de este tipo que separa cualquier solución real de las soluciones enteras. Actualmente se está llevando una gran investigación en este campo para distintos tipos de problemas de optimización bajo el *framework* "*Polyhedral combinatorics*" de Aardal y Weismantel, 1997.

Algoritmos: El método de Branch & Cut

Usando los dos métodos anteriores (tres sin contamos Simplex que ambos usan) llegamos al método de "*Branch & Cut*". Este consiste únicamente en aplicar Planos Cortantes hasta que no se consigue encontrar ningún plano que constriña más la solución. En el espacio resultante se usa entonces "*Branch & Bound*" para hallarla. Este es el principal algoritmo que usa CPLEX (Con muchas más optimizaciones, por supuesto) para hallar la solución de un problema MILP.

Paralelismo

Como se mencionaba en la introducción de este documento, la finalidad de la aplicación es realizar estudios de sensibilidad de los modelos en función de un parámetro que podrá elegir el usuario. Si definimos la granularidad de ejecutar los modelos para cada uno de los parámetros por separado como la cantidad de computo que se puede hacer sin tener que transmitir información de uno a otro, observamos que la granularidad de este problema es completa. Por tanto, deberíamos tratar de explotar esto lo más posible y la mejor forma de hacerlo es realizar los cálculos simultáneamente. Para esto vamos a recurrir a una serie de conceptos que definimos a continuación.

Clúster de computadores

Un clúster de computadores es un conjunto de computadores interconectados entre sí de tal manera que a efectos de realizar ciertas tareas actúan como uno solo. Estos computadores ejecutan cada uno su propia instancia de un sistema operativo.

Debido al relativo bajo coste de un procesador de propósito general y al aumento de la velocidad de las redes también a un reducido coste, un clúster es normalmente más eficiente con respecto al costo que un único computador con la misma capacidad y velocidad.

Para aprovechar al máximo la capacidad de cálculo de cada uno de ellos, tiene que existir un nodo que controle al resto o al menos una capa intermedia de *middleware* desde la que se gestione la carga de cada uno de los nodos.

Computación en la Nube

"La nube" es uno de los términos más en boga últimamente. Nos referimos a la nube como una infraestructura compleja que ofrece capacidad de cálculo y almacenamiento como un servicio a una serie de usuarios heterogéneos. Estos usuarios acceden a la nube a través de navegadores web o de ligeras aplicaciones de escritorio (Como la que se usa en este proyecto) o aplicaciones móviles. Según su despliegue podemos hablar de varios tipos de nubes:

- Publicas: El almacenamiento, las aplicaciones y los servicios son ofrecidas al público general por un proveedor de servicio.
- De comunidad: Son compartidas por una serie de organizaciones con preocupaciones comunes (Mismo sector, necesidades de seguridad, etc)
- Privada: Una única organización opera la nube, normalmente una empresa que ofrece los servicios a sus empleados.
- Híbrida: Incluye dos o más tipos de las anteriores entrelazadas.

Debido a la cantidad de secretos industriales que entran en juego en este proyecto, se pueden barajar dos posibilidades: La primera sería utilizar una nube privada, gestionada por la empresa que utilice la aplicación (En este caso, Enagás). La segunda sería colocarla en una nube pública que ofreciese suficiente seguridad como para poder realizar el envío de mensajes sin que estos fuesen interceptados y descodificados. Aunque la primera parece, a priori, mejor poder externalizar la gestión de la infraestructura que da apoyo a la aplicación (Como en el caso de colocarla en un servidor externo de *Amazon*, por ejemplo) es algo deseable y que a la larga provoca grandes ahorros económicos. Esto se debe a que en general, esta clase de nubes públicas se pagan por uso: solo se pagaría por los recursos utilizados en el momento en el que se realiza el análisis, sin necesidad de realizar mantenimiento de ningún tipo si ocurre lo contrario. Se entrará más afondo en las utilizaciones en esta clase de nubes públicas en el capítulo 4: Posibles ampliaciones.

Sobre los modelos de servicio que ofrece la nube, existen tres niveles: Infraestructura (Ofrecer máquinas, virtuales o no), plataforma (Ofrecer bases de datos, sistemas operativos, etc) o software (Aplicaciones concretas). El segundo es la que más nos interesa. El modelo de Plataforma Como Servicio (*Platform as a Service* o PaaS) permite al usuario crear software y colocarlo en el proveedor, que ofrece la red, los servidores y el almacenamiento.

Servicios Web

Un servicio web es un sistema software diseñado para ofrecer interacción entre ordenadores. Esto se realiza a base de pares petición-respuesta codificados como mensajes de texto en formato .xml.

Un servicio web posee una interfaz en WSDL (del inglés *Web Service Description Language* o Lenguaje de Descripción de Servicios Web) que no es más que un lenguaje basado en XML que puede ser leído por una máquina y que describe de que forma puede ser llamado un servicio web, que parámetros admite y que estructuras de datos devuelve. La estructura básica de un fichero en WSDL consta de los siguientes apartados

- <types>: Tipos de datos usados en los mensajes
- <message>: Elementos del mensaje cada uno de un tipo definido en la parte anterior
- <portType> Operaciones permitidas y mensajes intercambiados
- <binding> Protocolos de comunicación usados

Gracias a esto, un servicio web es idóneo para intercambiar datos entre aplicaciones, independientemente del lenguaje en que estén programadas y de las máquinas en las que se encuentren las mismas.

Otra de las principales ventajas de los servicios web es que en gran medida son independientes de las aplicaciones que los consumen, lo que hace que a la hora de realizar modificaciones en uno de los dos extremos –aplicación o servicio- no tengamos que realizar ajustes en el otro. Dada la repercusión que tiene hoy día el paradigma de programación orientada a componentes, se cree que esta flexibilidad aportada por los servicios web los ayudará a tener mucha presencia en adelante y revertirá en un mayor uso de estos, aumentándose la cantidad y calidad de los servicios web.

Sin embargo, no están exentos de algunos inconvenientes importantes. Entre estos encontramos por un lado un pobre desarrollo de transaccionabilidad y de rendimiento de procesamiento si se compara con otros modelos de programación distribuida. El problema del rendimiento viene intrínsecamente ligado a la propia naturaleza del servicio, dado que está basado en XML. La otra gran desventaja es de seguridad, y es que al basarse en HTTP, pueden transgredir las reglas de los firewall.

Llamada a procedimiento remoto

Un servicio web puede ser usado de varias formas diferentes. Para este proyecto en particular nos interesa el de “Llamada a procedimiento remoto” (*Remote Procedure Call* o RPC) Este es un proceso de comunicación que provoca que una subrutina sea ejecutada en espacio de direcciones dife-

rente (en nuestro caso, una máquina diferente) y que provee la forma de hacerlo sin tener que administrar directamente la tarea, únicamente haciendo una llamada a subrutina como si se tratase de la misma máquina. Esto sigue un paradigma cliente-servidor, siendo el cliente la máquina que realiza la petición y el servidor la que ejecuta el método y devuelve los resultados de dicha ejecución. Durante una invocación a un procedimiento remoto se realizan una serie de pasos

- El cliente ejecuta localmente un segmento de código (stub) que adecua los parámetros para realizar la llamada (Por ejemplo, los punteros tienen que ser convertidos ya que no tienen sentido en el espacio de memoria de la otra máquina).
- Este mismo segmento de código empaqueta los parámetros en un mensaje y hace una llamada al sistema para enviar dicho mensaje. Este empaquetamiento se conoce como "*marshalling*" y es similar a la serialización.
- El sistema operativo del cliente envía el mensaje a la máquina que actúa como servidor
- El sistema operativo del servidor pasa los parámetros al "stub" del mismo
- Este último hace la llamada al procedimiento.

La respuesta se envía al cliente siguiente la secuencia inversa.

SOAP

Los mensajes que se describen en el apartado anterior requieren enviarse bajo cierto protocolo para que puedan transitar por la red y recibirse correctamente en su destino. El protocolo que usamos para ello se denomina SOAP (*Simple Object Access Protocol* o Protocolo de Acceso a Objeto Simple) Este es una especificación para el intercambio de información estructurada (Básicamente, archivos .xml) en el marco de un servicio web. Usualmente utiliza otro protocolo de la capa de Aplicación del modelo OSI, siendo el más notable de ellos HTTP.

SOAP proporciona un *framework* de mensajería sobre el que se puede construir el servicio web. Consta de tres partes:

- Un envoltorio, que define que es un mensaje y como procesarlo.
- Unas reglas de codificación para expresar las instancias de tipos de datos definidos por las aplicaciones.
- Un convenio para representar las llamadas a procedimientos y las respuestas de estos.

Así mismo, un mensaje SOAP está estructurado también en tres partes: Envoltorio, cabecera y contenido.

Como se ha comentado, SOAP puede utilizar HTTP como protocolo de la capa de aplicación para transporte. Además de esto, se puede utilizar HTTPS, lo que es muy relevante para la aplicación ya que los datos que se envían bien podrían declararse secreto industrial.

La Pila de Protocolos del Servicio Web

Una vez definidos todos los conceptos y protocolos anteriores vamos a ver como se estructuran dentro del servicio web. Esta estructura sigue un esquema de capas o bloques, apilados uno encima del otro en lo que se llama la Pila de Protocolos del Servicio Web. Esta incluye cuatro capas, de arriba a abajo en la pila:

- Protocolo de transporte: Se encarga de llevar los mensajes entre aplicaciones basadas en red e incluye protocolos como el citado HTTP, SMTP o FTP.
- Protocolo de mensajería: Se encarga de codificar los mensajes como XML para que puedan ser leídos por ambas partes de la conexión. Actualmente los más usados son XML-RPC, WS-Adressing y SOAP.
- Protocolo de descripción: Se usa para describir la interfaz pública del servicio web. El lenguaje WSDL es el estándar para este propósito.
- Protocolo de descubrimiento: Centraliza un registro común de servicios web a través de la red de forma que sea más fácil descubrir que servicios están disponibles. Es de menor interés para este proyecto.

Estructura de la pila de protocolos en nuestro proyecto

Capítulo 2: La Aplicación

Introducción al Capítulo

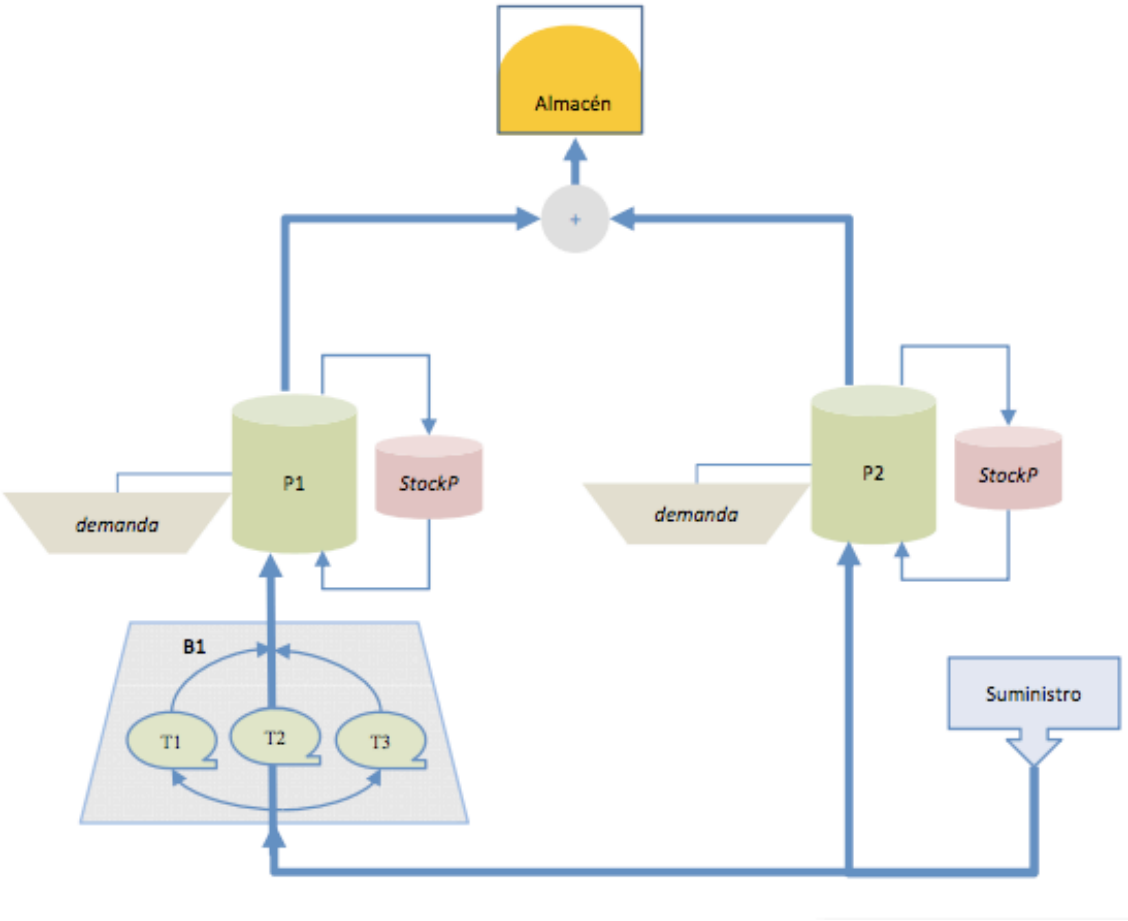
En este capítulo vamos a pasar a describir en detalle que consiste y como funciona la aplicación, utilizando los conceptos definidos en el capítulo anterior. Empezaremos definiendo como se modela el sistema con un ejemplo de pequeño tamaño y que utilizaremos a lo largo del capítulo como caso de estudio para ver como se resolvería un problema real.

Más adelante veremos que método se ha utilizado para aislar al usuario de la programación por restricciones y de CPLEX, de tal forma que los conocimientos sobre las redes logísticas de gas sean lo único necesario para poder utilizar la aplicación de una forma totalmente funcional.

Por último se explicará como explotar el paralelismo inherente a los análisis de sensibilidad utilizando servicios web y computación distribuida., englobándolo en el uso de la aplicación.

Modelado de la Red

Como se explicó en el apartado dedicado a CPLEX, un problema consta de dos ficheros: uno que contiene el modelo propiamente dicho y otro que incluye los datos a ejecutar con ese modelo. Vamos a introducir un sistema sencillo y veamos como se modela:



Ejemplo de modelo
(Ilustración perteneciente al Máster en Ingeniería de Sistemas y de Control, usado con autorización)

Como se puede ver, este es un sistema que tiene un único punto de suministro pero con dos salidas, una va directamente a una estación de compresión y otra a un almacén a través de una tubería. Por el otro extremo de la bomba sale a su vez otra tubería que se conecta a la anterior y al almacén, cerrando el circuito. Cada tubería tiene un punto de demanda que hay que satisfacer para que el sistema tenga solución. Desglosemos el fichero que modelaría este sistema.

Primero, hay que definir los parámetros que afectan a todo el sistema: El número de ciclos de ejecución y dos rangos que se usarán como tamaño de las matrices de datos. Los puntos suspensivos en una variable indican que no está definida y que por tanto debe introducirse un valor en el fichero de datos.

```
//DATOS
int Periodo= ...;          //Numero de ciclos
range T = 1.. Periodo ;    //Rango sin condiciones iniciales
range T0 = 0.. Periodo ;   //Rango con condiciones iniciales
```

A continuación, para cada clase de elemento del sistema hay que definir los datos que va a utilizar y que son los que definirá el usuario:

```
//Modelo de la estación de compresión (Bidireccional) Datos
int numTurbos = ...;       //Posiciones de funcionamiento de las bombas;
{string} bombas = ...;     //Nombres de las estaciones;
range turbos = 1..numTurbos; //Rango de posiciones de funcionamiento;
float fMin[bombas,turbos] =...;
     //Flujo mínimo en función de la posición y la estación;
float fMax[bombas,turbos] =...;
     //Flujo máximo en función de la posición y la estación;
float consumo[bombas,turbos] = ...;
     //Consumo en función de la posición y la estación;
```

```
//Modelo de almacenamiento (Bidireccional) Datos
{string} almacenes = ...;      //Nombres de los almacenes
int cuota [almacenes]= ...;    //Capacidad de entrada/salida
float stockInitA[almacenes]= ...;  //Estado inicial
```

```
//Modelo de conducción (Bidireccional) Datos
{string} pipes = ...;         //Nombres de las tuberías
float stockInitP[pipes]= ...;//Estado inicial de las tuberías
float stockMinP[pipes]= ...; //Capacidad mínima de las tuberías
float stockMaxP[pipes]= ...; //Capacidad máxima de las tuberías
float demandaP[T,pipes]= ...;//Demanda de salida por tubería e instante
```

```
//Modelo de suministro (Planta) Datos
{string} plantas = ...;  //Nombre de las plantas
float suministro[T,plantas] = ...;//Producción de la planta por periodo
```

Una vez hecho esto, hay que definir las variables que se pretende optimizar. Estas serían las variables que se dejan "libres" y las que CPLEX manipula para alcanzar el óptimo. Igual que en el apartado anterior, cada clase de elemento tiene unas variables:

```
//Modelo de la estación de compresión (Bidireccional) Variables
dvar int dp[T,bombas,turbos] in 0..1; //Flujo positivo?
dvar int dn[T,bombas,turbos] in 0..1; //Flujo negativo?
dvar float FlujoBP[T,bombas];  //Flujo en dirección positiva
dvar float FlujoBN[T,bombas];  //Flujo en dirección negativa
dvar float+ FlujoB[T,bombas];  //Flujo total en la bomba
```

```
//Modelo de almacenamiento (Bidireccional) Variables
dvar float FlujoA[T,almacenes];   //Flujo que entra/sale del almacén
dvar float+ StockA[T0,almacenes]; //Cantidad que queda en el almacén
```

```
//Modelo de conducción (Bidireccional) Variables
dvar float FlujoPE[T,pipes];  //Flujo de entrada de la tubería
dvar float FlujoPS[T,pipes];  //Flujo de salida de la tubería
dvar float StockP[T0,pipes];  //Cantidad que hay en la tubería
```

```
//Modelo de suministro (Planta) Variables
dvar float+ suministroP[T,plantas];
```

La función objetivo del problema se coloca a continuación

```
//Función objetivo (Minimizar el consumo de las estaciones de compr.)
minimize sum(i in T,j in bombas, k in turbos)
                (dp[i,j,k]+dn[i,j,k])*consumo[j,k];
```

Definidas todas las variables que intervienen en el problema, toca el turno para las restricciones que las relacionan entre sí. Una vez más, se agrupan por tipo de elemento. Nótese que

```
subject to
{
//Modelo de la estación de compresión (Bidireccional) Restricciones
forall(t in T, j in bombas)
{
        sum(k in turbos)dp[t,j,k]*fMin[j,k] <= FlujoBP[t,j];
//El flujo positivo es mayor que el minimo de funcionamiento de la bomba
        sum(k in turbos)dp[t,j,k]*fMax[j,k] >= FlujoBP[t,j];
//El flujo positivo es menor que el maximo de funcionamiento de la bomba
        sum(k in turbos)dn[t,j,k]*(-fMin[j,k]) >= FlujoBN[t,j];
//El flujo negativo es mayor que el minimo de funcionamiento de la bomba
        sum(k in turbos)dn[t,j,k]*(-fMax[j,k]) <= FlujoBN[t,j];
//El flujo negativo es menor que el maximo de funcionamiento de la bomba
        sum(k in turbos)(dp[t,j,k]+dn[t,j,k])==1;
//Solo se puede bombear en un sentido en un momento dado
         FlujoB[t,j] == FlujoBP[t,j]+FlujoBN[t,j];
//El flujo total es la suma del flujo positivo y del negativo
}
```

todas estas restricciones deben ir encapsuladas en un bloque "subject to {...}"

```
forall(j in almacenes)
        StockA[0,j] == stockInitA[j]; //Estado inicial de los almacenes
forall(t in T, j in almacenes)
{
     StockA[t,j]==StockA[t-1,j] + FlujoA[t,j];
          //El estado del almacén en el momento t es el del momento
          t-1 + el flujo de entrada/salida
      -cuota[j] <=FlujoA[t,j]<= cuota[j]
          //El flujo de entrada/salida esta dentro del rango permitido
}
```

```
//Modelo de conducción (Bidireccional) Restricciones
forall(i in pipes)
   StockP[0,i] == stockInitP[i]; //Estado inicial de las tuberías
forall(t in T, j in pipes)
{
    FlujoPS[t,j]== FlujoPE[t,j]+StockP[t-1,j]-StockP[t,j]-demandaP[t,j];
//El flujo de salida en el instante t es el flujo de entrada en ese
//instante, más la cantidad de de gas que había en la tubería en el
//instante t-1, menos la cantidad de gas que queda en la tubería,
//menos la demanda que ha salido de la tubería al exterior
    StockP[t,j] <= stockMaxP[j];
//La cantidad de gas que transporta la tubería no excede sus limites
//estructurales
    StockP[t,j] >= stockMinP[j];
//La cantidad de gas que transporta la tubería está por encima del
//mínimo de funcionamiento
}
```

Finalmente hay que añadir las restricciones que definen realmente el modelo. Todo lo anterior solo afecta a la forma en la que se definen los distintos componentes del modelo y, llegado el caso podrían utilizarse tal cual aparecen para cualquier modelo en el que no se quisiesen introducir más parámetros. (Por ejemplo, si se decidiese que para una orografía particular el factor de inclinación de

las tuberías es importante y debiera tenerse en cuenta, habría que modificar el modelo de la tubería y sus restricciones de forma adecuada) En cualquier caso, la topografía concreta de un modelo se define en esta parte como un conjunto de restricciones del flujo. Estas fuerzan que en todos los nodos de la red el flujo combinado sea igual a cero o lo que es lo mismo, que el gas no se fuga del sistema ni se inyecta más allá del punto de suministro. Para el modelo que aparece al comienzo de esta sección estas restricciones serían:

```
forall(t in T)
{
        FlujoPE[t,"P1"]== FlujoB[t, "B1"];
        FlujoPS[t,"P1"]+FlujoPS[t,"P2"]== FlujoA[t,j];
        FlujoB[t,"B1"]+FlujoPE[t,"P2"]== suministro[t,"PL1"];
}

}//Fin de las restricciones
}//Fin del programa
```

La primera de ellas indica que el flujo de entrada en la tubería uno es el flujo de salida de la estación de compresión, la segunda que el flujo de salida de la tubería uno más el flujo de salida de la tubería dos es la cantidad de gas que entra en el almacén y la tercera que el flujo de entrada en la bomba más el flujo de entrada en la tubería dos debe ser igual a la cantidad que entra en el sistema.

Con esto queda definido un modelo que deberá estar alojado en cada uno de los servidores para realizar el cálculo. Comentemos ahora como queda definido el fichero de datos que permite ejecutarlo. Este es sustancialmente más corto y sencillo que el modelo, como se puede ver a continuación:

```
//DATOS
Periodo = 3;                //Numero de ciclos
//-------------------------------------------------------------------------
//Modelo de la estación de compresión (Bidireccional) Datos
numTurbos=3;          //Posiciones de funcionamiento de las bombas;
bombas = {"B1"};      //Nombres de las estaciones;
fMin = [[10.1, 20.2,30.3]];
     //Flujo mínimo en función de la posición y la estación;
fMax = [[15.1, 25.2,35.3]];
     //Flujo máximo en función de la posición y la estación;
consumo = [[15.1, 25.2,35.3]];
     //Consumo en función de la posición y la estación;
//-------------------------------------------------------------------------
//Modelo de almacenamiento (Bidireccional) Datos
almacenes = {"A1"}; //Nombres de los almacenes
cuota= [40];        //Capacidad de entrada/salida
stockInitA= [0];    //Estado inicial
//-------------------------------------------------------------------------
//Modelo de conducción (Bidireccional) Datos
pipes = {"P1","P2"};//Nombres de las tuberías
stockInitP= [0,0];  //Estado inicial de las tuberías
stockMinP= [0,0];   //Capacidad mínima de las tuberías
stockMaxP= [10,10]; //Capacidad máxima de las tuberías
demandaP= [[40,30],[30,15],[10,20]];
     //Demanda de salida en cada tubería e instante
//-------------------------------------------------------------------------
//Modelo de suministro (Planta) Datos
plantas = {"PL1"};  //Nombre de la planta
suministro = [[120],[180],[90]]; //Producción por periodo
```

Como se puede ver, se trata de rellenar los valores que se dejaron sin rellenar en el modelo,prestando atención a utilizar los mismos nombres para las variables y los mismos tipos de datos ya definidos.

Creación del fichero de datos

Aunque el fichero con el modelo sin duda tiene que ser creado por alguien que conozca como funciona la red y que tenga conocimientos de OPL y CPLEX, no ocurre lo mismo con el fichero de datos. Cualquier personas que pueda tener acceso a la distribución de la red de gas debería poder utilizar la aplicación sin conocimientos previos de ningún lenguaje de programación. Por tanto se ha añadido una capa intermedia de comunicación con el usuario que permite leer los datos directamente de un fichero de Microsoft Excel. Se ha escogido este formato por su facilidad de uso y su amplia extensión. Aunque se proporciona una plantilla con la aplicación, debido a que el usuario debe conocer el modelo, él mismo puede crear uno de estos ficheros y rellenarlo con los datos que desee. A continuación se muestra uno de estos ficheros y que representa cada una de sus partes:

	A	B	C	D	E	F	G	H	I	J
1		Número de Periodos de la simulación								
2	Periodo	3								
3		Número de posiciones de funcionamiento								
4	numTurbos	3								
5		Nombres de las bombas								
6	bombas	B1								
7		Flujos mínimo, máximo y consumo en función de la posición de funcionamiento y de la bomba								
8	fMin	10.1	20.2	30.3						
9	fMax	15.1	25.2	35.3						
10	consumo	15.1	25.2	35.3						
11		Nombres de los tanques								
12	almacenes	A1								
13		Capacidad de E/S de cada tanque								
14	cuota	40								
15		Stock inicial de cada tanque								
16	stockInitA	0								
17		Nombres de las tuberias								
18	pipes	P1	P2							
19		Stock inicial, mínimo, máximo por tubería								
20	stockInitP	0	0							
21	stockMinP	0	0							
22	stockMaxP	10	10							
23		Demanda en cada tubería por periodo								
24	demandaP	50	70	#	70	70	#	10	20	
25		Nombres de las plantas								
26	plantas	PL1								
27		Output de las plantas por periodo								
28	suministro	120	#	180	#	90				
29	Parámetros a variar									
30	consumo	20	40	60						
31	@									
32	demandaP	50	50	#	70	70	#	15	60	

Las filas coloreadas son meramente informativas pero deben colocarse antes de cada dato para que el archivo sea correctamente procesado. La columna de la izquierda contiene los nombres de los distintos elementos del modelo. Por ejemplo, en este modelo la matriz que contiene la demanda por tubería y periodo se llama "demandaP". A su vez, las matrices se definen mediante secuencias de vectores en los cuales cada línea va separada de la siguiente por una almohadilla (#). Las celdas en blanco no cuentan a la hora de procesar el fichero. Esto facilita la lectura ya que, si por ejemplo este modelo tuviese una segunda estación de compresión, sus valores de flujos mínimo, máximo y consumo se escribirían en la fila adecuada a partir de la columna cinco. Podríamos dejar las columnas dos, tres y cuatro de la fila nombre en blanco y escribir el nombre de la estación 2 en la cinco de tal forma que quedase alineada con sus datos.

Sin embargo la parte más importante del fichero es la sección de "Parámetros a variar". Entre el título y la marca de fin de fichero se pueden repetir los nombres de las partes del modelo y un valor alternativo de los datos que se ejecutarán en un servidor diferente. Cada una de estas alternativas se debe separar mediante una línea con la marca "@". Esto se explicará con detalle en la sección "Paralelismo" de este capítulo. Por ahora baste decir que por cada valor extra que se introduzca en esta parte del fichero, se ejecuta una instancia del problema con todos los datos iniciales excepto el marcado que es el que varía. Por ejemplo, para la imagen que aparece más arriba, se ejecutaría el modelo en un servidor con todos los parámetros que aparecen, incluyendo un flujo mínimo de las estaciones de 10.1 unidades en la posición de la bomba 1, 20,2 en la posición 2 y 30,3 en la posición 3. A su vez, en otro servidor se ejecutaría este mismo modelo con los mismos parámetros, a excepción del flujo mínimo que valdría 20, 23 y 56 respectivamente para cada una de las citadas posiciones.

Utilizando este formato, no solo se facilita una mayor accesibilidad a la aplicación sino también se limitan los potenciales errores que se pueden cometer al fichero de datos. Se debe resaltar que el modelo que hemos mostrado no es más que un pequeño ejemplo, un modelo real podría tener varias plantas de suministro, docenas de estaciones de compresión y almacenes y centenares de tuberías. Esto implicaría un fichero de datos que podría tener varios cientos de líneas de código. Si suponemos que el usuario de la aplicación no dispone del entorno de desarrollo gráfico de CPLEX (Como sería lo normal, si no podría realizar los análisis localmente sin necesidad de recurrir a la red) la depuración de posibles errores de escritura se convierte en una tarea farragosa. Por el contrario, introducir únicamente valores numéricos en la hoja excel resulta mucho más sencillo y visualmente mucho más fácil de comprobar si se detecta que algo no funciona como debería.

Como la hoja de cálculo que crea el fichero .dat está íntimamente ligada al fichero con el modelo, las dos se almacenan juntas en las máquinas que posean CPLEX. Si un usuario desea hacer un análisis de sensibilidad, puede realizar una petición a una de estas máquinas, obteniendo como resultado la plantilla adecuada para el modelo totalmente vacía. Una vez más, esto permite que casi cualquier usuario pueda utilizar la aplicación sin ningún conocimiento previo a parte de saber como está estructurado el modelo. En un principio se pensó en enviar el modelo junto con el fichero excel con los datos al servidor en forma de paquete. Sin embargo, se desechó está idea debido a temas de licencias del resolutor (Veáse Anexo: Licencias).

Paralelismo

El usuario ya tiene listo su fichero con los datos a ejecutar. Es más, ya ha añadido las filas extra que incluyen los parámetros de su análisis de sensibilidad. El siguiente paso es enviar dicho fichero a una máquina que disponga de CPLEX y que pueda resolver el problema empleando el mínimo tiempo posible. Debido a la granularidad completa del mismo. (Ver capítulo 1: Paralelismo) lo ideal es hacer uso de tantas máquinas como valores del parámetro tengamos y ejecutar cada una de estas instancias en una máquina (a partir de ahora, servidor) diferente. El fichero debe guardarse en formato .xml (Cosa que permite Microsoft Excel) de tal forma que sea más sencilla su decodificación.

Para que esto pueda realizarse, los servidores deben estar conectados a la red y esperando peticiones. Por tanto, cada servidor estará ejecutando una instancia del servicio web de la aplicación. Este, junto con la aplicación de escritorio que dispone el cliente, es una de las dos partes que componen la misma y es, sin duda, la más importante. El cliente debe sin embargo saber en que dirección puede localizar a los servidores para enviar los datos. Antes de realizar ningún otro cálculo, se leen desde un fichero de configuración las direcciones de los servidores. A continuación se ejecuta un pequeño programa de prueba para comprobar que efectivamente se dispone de estos servidores. En caso de fallar esta fase, se notificaría al usuario para que redujese el número de valores del análisis de sensibilidad. La segunda motivación de este programa de prueba es obtener el tiempo que tarda en ejecutarse para así priorizar unas máquinas sobre otras. Una máquina excepcionalmente lenta o que ya esté soportando una carga de trabajo muy alta degradará en exceso el rendimiento de todo el análisis de sensibilidad. Por tanto se distribuye la carga de trabajo adecuadamente entre los servidores dependiendo de su velocidad.

Por tanto, la aplicación se estructura como un clúster, en el que el cliente es la máquina que utilizará el usuario y que está conectado con una serie de servidores en los que se han desplegado distintas instancias del servicio web. El cliente dispondrá de una lista de direcciones en las que conoce que se encuentran estos servidores y realizará una llamada a cada uno de los mismos para que ejecuten remotamente el procedimiento que resuelve el modelo. Para poder realizar esta tarea paralelamente, se crean tantos threads en la máquina cliente como servidores existan y se hace cada llamada en uno de los threads. La finalidad de esto último es que el cliente no tenga que esperar la respuesta de cada servidor para poder lanzar la petición al siguiente, si no que se realicen a la vez.

La diferencia que existen entre los mensajes que se envían a cada servidor radica en que estos mensajes implicarán que en cada servidor se ejecute el modelo para un valor diferente del parámetro seleccionado. Por tanto, los servidor ejecutarán el mismo modelo pero con distintos datos paralelamente e idealmente el análisis de sensibilidad se realizará en 1/n del tiempo que se tardaría en hacerlo secuencialmente. Este escenario ideal solo se presentará si todos los servidores son capaces de resolver el modelo con la misma rapidez y si el envío de mensajes entre estos y el cliente tarda siempre el mismo tiempo, pero en cualquier caso se consigue un grado de paralelismo total. Esto es debido como ya se ha explicado a que la granularidad del problema es absoluta: cada servidor no necesita realizar ninguna comunicación con otro servidor ni con el cliente una vez ha empezado el proceso de resolución que no sea devolver los resultados obtenidos.

Estructura de la aplicación

El fichero de datos está en formato Excel como se ha dicho. Sin embargo, la conversión a .xml es inmediata, quedando estructuradas las celdas como un árbol de etiquetas. Esta es la segunda razón por la que se utilizó el formato Excel. Ya que el servicio web utiliza el SOAP como protocolo de mensajería, además de añadir el envoltorio y la cabecera necesarias, debe hacer una transformación de los datos del mensaje a formato .xml para poder enviarlos. Si los datos ya se encuentran efectivamente en ese formato, toda la tarea se realiza más rápidamente. Especialmente si se da el caso, como sería lo usual en si la aplicación estuviese desplegada en una empresa, de que el fichero de datos incluye cientos de parámetros para el sistema.

Otra idea importante a destacar es que en el momento que se hacen las peticiones en distintos threads, a cada uno de estos se le asigna un número secuencial. Este número se envía en la petición a cada servidor e identifica unívocamente a cada uno de ellos. La importancia de esto radica en que la tabla de datos se envía completa, esto es con toda la información de paralelismo íntegra en formato .xml y es en cada servidor donde se construye efectivamente el fichero de datos que usará el resolutor. Por tanto es necesario que durante este proceso de construcción se conozca cual de las filas que redefinen parámetros ha de usarse.

Una vez que se tienen los datos con la línea correcta que se debe utilizar en el servidor, se reconstruye un fichero de texto plano, que es el que CPLEX necesita para operar. Este fichero se guarda en la carpeta de almacenamiento temporal del servicio web y tiene una apariencia similar al que se muestra en la página 39 de esta memoria. Desde aquí, el procedimiento remoto realiza una llamada a la biblioteca ILOG del CPLEX que incluye tanto el fichero del modelo como el recién creado fichero de datos y se inicia el procedimiento de resolución con los algoritmos citados en el capítulo anterior.

CPLEX se utiliza en la aplicación escrita en C# por medio de las bibliotecas CPLEX e ILOG, que quedan de esta forma incluidas en el proyecto. Estas ofrecen toda una serie de funcionalidad que puede ser aprovechada para analizar la ejecución del modelo. A la hora de devolver los resultados, se añaden al fichero de datos una serie de líneas que permiten escribir los valores de las variables del sistema en su punto óptimo directamente en la hoja de cálculo que ha recibido el servidor. Recordemos que el principal interés a la hora de planificar es tanto conocer cual es dicho óptimo como saber que pasos hay que llevar a cabo para conseguirlo. En un sistema logístico de gas esto se traduce en a que régimen debemos poner a trabajar una estación de compresión en cada instante. Además nos interesan también otra serie de variables que nos proporcionan información sobre la cantidad de gas que circula por las tuberías, el estado de los almacenes en cada instante, etc.

Como hemos dicho estos valores necesarios para la planificación se escriben en la hoja de cálculo mediante sentencias en el fichero .dat. Estas son del estilo:

```
SheetConnection sheet("C:/TransientStorage/libro1.xml");
FlujoPE to SheetWrite(sheet,"RESULTADOS!D20:E22");
```

donde la primera sentencia crea la conexión entre CPLEX y el fichero Excel y la segunda es un ejemplo en el que se ordena que la variable "FlujoPE" que contiene el flujo de entrada en las tuberías en cada instante se escriba como una matriz en las celdas D20 a la E22 (Matriz de 2x3) en el libro resultados.

El servidor tras resolver el modelo y escribir los datos, devuelve de nuevo el fichero Excel, marcado de tal forma que se sepa en que servidor y con que datos concretos se realizó el cálculo. De esta forma el usuario final de la aplicación tiene de un vistazo no solo que solución es mejor, si no también como se comporta el sistema internamente para cada una de ellas.

	A	B	C	D	E	F	G	H	I	J	K
1											
2		Flujo por Bomba y Periodo									
3			**Bombas**	**B1**							
4		**Periodos**									
5		**1**		15,1							
6		**2**		15,1							
7		**3**		15,1							
8											
9		Cantidad de Gas por Tanque y Periodo									
10			**Tanques**	**A1**							
11		**Periodos**									
12		**0**		0							
13		**1**		0							
14		**2**		40							
15		**3**		80							
16											
17		Flujo en Dirección Positiva por Tubería y Periodo									
18			**Tuberías**	**P1**	**P2**						
19		**Periodos**									
20		**1**		15,1	104,9						
21		**2**		15,1	164,9						
22		**3**		15,1	74,9						
23											
24		Flujo en Dirección Negativa por Tubería y Periodo									
25			**Tuberías**	**P1**	**P2**						
26		**Periodos**									
27		**1**		-34,9	34,9						
28		**2**		-54,9	94,9						
29		**3**		-4,9	44,9						
30											
31		Stock en Cada Tubería Por Periodo									
32			**Tuberías**	**P1**	**P2**						
33		**Periodos**									
34		**0**		0	0						
35		**1**		0	0						
36		**2**		0	0						
37		**3**		10	10						

Con estos datos el usuario puede realizar la planificación del sistema. Por ejemplo, y para el sistema de prueba con el que hemos estado trabajando, habría que colocar la única estación de compresión del sistema en la posición de funcionamiento uno (La que mueve una cantidad de gas de 15.1 unidades por periodo) durante los tres periodos, obteniéndose un consumo de 45.3 unidades durante todo el periodo de planificación. La demanda quedaría plenamente satisfecha. El valor óptimo y la instancia concreta de los datos con los que se ejecutó el modelo pueden consultarse también como se ve en la siguiente imagen.

	A	B	C	D	E	F	G
1	OPTIMO:	**950**		**Para la ejecución con fMin[[10,20,40]]**			
2		**Flujo por Bomba y Periodo**					

Capítulo 3: Desarrollo

Introducción al capítulo

Aunque gran parte del esfuerzo de este proyecto ha corrido a cargo de la investigación de todo lo relacionado con los apartados anteriores (Desde como modelar una red de transporte de gas a cómo utilizar CPLEX, pasando por construir servicios web y explotar el paralelismo a través de los mismos) no podemos de dejar de hablar del trabajo que se ha invertido en el desarrollo de la misma. Vamos a pasar a describir el entorno de desarrollo que se ha usado en este proyecto, el lenguaje usado y los retos afrontados. Seguidamente comentaremos la implementación de partes críticas de la aplicación, ilustrándolas con código si es necesario.

Visual Studio 2010

Sobre Visual Studio

Microsoft Visual Studio ha sido, tanto a lo largo de la última década como de la presente, el entorno de desarrollo de referencia para todos aquellos desarrolladores de aplicaciones basadas en sistemas operativos Windows (Microsoft Windows, Windows CE o Windows Mobile, entre otros). Su fiabilidad y robustez a la hora de diseñar, programar, desplegar y depurar aplicaciones son, junto con su versatilidad, el punto fuerte de este paquete integrado.

Visual Studio se pronuncia como uno de los paquetes más versátiles, debido fundamentalmente al amplio soporte que ofrece para numerosos lenguajes de programación, tales como Visual Basic, Visual J#, ASP.NET, C++ ó C#. Otros lenguajes como Python y Ruby también son soportados mediante la instalación de módulos y extensiones adicionales. Además, soporta lenguajes de marcado y estilo tales como XML, CSS o HTML.

El paquete contiene un avanzado editor de código y un depurador, así como un diseñador web. Además, cuenta con un completo editor de estructuras de Bases de Datos.

Esta herramienta no sólo se limita a ofrecer un soporte seguro para crear aplicaciones para entornos Windows. Va más allá, permitiendo a los desarrolladores programar sitios y servicios web, así como aplicaciones para móviles. Con este propósito, Microsoft ha procurado facilitar la tarea de intercomunicar todo este tipo de aplicaciones a través de un framework creado para tal fin en el año 2002, como es .NET, sobre el cuál haremos hincapié más adelante.

Visual Studio 2010

Como consecuencia del énfasis puesto por Microsoft en facilitar el desarrollo e implementación de paquetes web, surge esta última versión del framework, lanzada el 12 de Abril de 2010.

Los puntos fuertes de esta versión son, sin duda alguna, la facilidad de desarrollo y la integración total con Windows 7, así como la simplificación de las tareas relacionadas con el desarrollo de módulos y servicios web.

Se añadió retrocompatibilidad con múltiples versiones anteriores de .NET (incluyendo la actual 4.0) y se consiguió agilizar la implementación de servidores web gracias a IntelliSense -una potente

herramienta de documentación e inserción de código-, pero el punto que verdaderamente hace falta destacar es el soporte que ofrece para poder interactuar de manera satisfactoria con los servidores IIS (*Internet Information Service*), que permiten publicar servicios web de manera sencilla.

Dada la naturaleza del presente proyecto, puramente basada en servicios web y orientada fundamentalmente a la recogida de parámetros de forma distribuida a través de diferentes servidores, se ha considerado oportuno el uso de la versión 2010 de Visual Studio, con el fin de agilizar la configuración y despliegue de dichos servicios web.

C#

Motivación

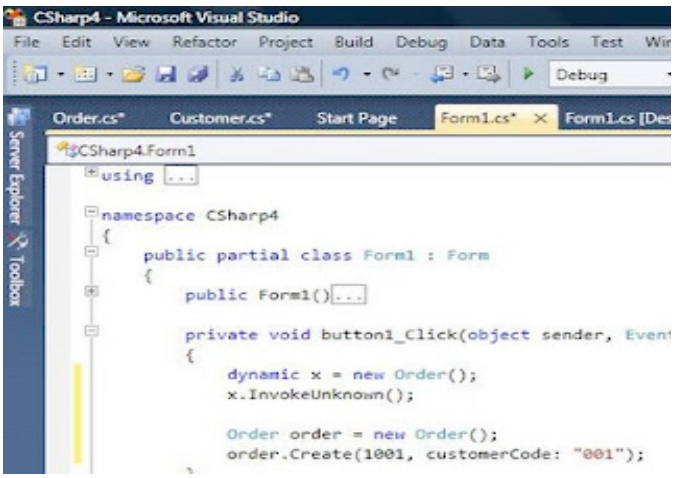
La utilización de CPLEX como una biblioteca adicional de Visual Studio ha motivado en el desarrollo de este proyecto el uso del lenguaje C#, a través del cuál se pueden hacer las llamadas hacia esta estructura. Aunque esto está soportado también en otros lenguajes, la relación entre C# y el framework .NET facilitaba mucho la creación de la parte web de la aplicación.

¿Qué es C#?

C# es un lenguaje de programación de propósito general, multiparadigma, imperativo, declarativo, fuertemente tipado y orientado a objetos.
Fue desarrollado y estandarizado por Microsoft en el año 2001 como parte de su, por entonces, nueva plataforma de desarrollo ".NET". Su creación vino motivada por la necesidad de disponer de un lenguaje diseñado desde cero y sin elementos heredados de pasadas versiones de C y otros lenguajes, sin elementos innecesarios que dificultase la utilización e interacción con .NET y que resultase lo más sencillo posible para poder aprovechar su versatilidad y potencia.

Este lenguaje combina los mejores elementos de varios de los lenguajes de mayor difusión como C++, Java o Visual Basic. De hecho, su creador Anders Heljsberg fue también el creador de muchos otros lenguajes y entornos como Turbo Pascal, Delphi o Visual J++. La idea principal detrás del lenguaje es combinar la potencia de lenguajes como C++ con la sencillez de lenguajes como Visual Basic, y que además la migración a este lenguaje por los programadores de C/C++/Java sea lo más inmediata posible.

La implementación más conocida de la especificación de este lenguaje es Visual C#, incluida en todos los productos de la gama Visual Studio.



Visual C# era un lenguaje hasta la fecha desconocido para el desarrollo de este proyecto. El énfasis puesto por los desarrolladores en hacer un lenguaje sencillo y fácilmente migrable desde Java o C++ se hizo patente a través de la rápida familiarización con la sintaxis, dadas las claras similitudes con los lenguajes en los cuales está basado C#.

Threads en C#

Situándonos dentro del contexto de esta aplicación, resulta esencial e imprescindible de cara a mejorar la escalabilidad del sistema, así como la velocidad de respuesta, la recogida de diferentes parámetros y restricciones de forma distribuida mediante el envío de múltiples ficheros de restricciones procedentes de diferentes servidores, e incluso aprovechar las computadoras actuales multinúcleo procesando un modelo por cada uno de los núcleos.

Frente a la problemática de procesar más de un archivo a la vez, surge la necesidad de utilizar *multithreading*. Mientras tengamos la capacidad de crear enésimos hilos, podremos ejecutar el envío de cada archivo de forma independiente garantizando la máxima granularidad del sistema y contribuyendo a mejorar el tiempo de respuesta de la aplicación como consecuencia de explotar este paralelismo.

C# nos propone una sencilla forma de manipular individualmente los hilos a través de los métodos y directivas definidos en la librería `System.Threading`. Cada hilo pertenecerá a una nueva clase en la cuál será necesario definir, aparte de métodos y atributos propios, las siguientes directivas:

- `DoWork()`: Este método será el invocado cuando el hilo sea iniciado.
- `RequestStop()`: Desde este método solicitaremos la parada de la ejecución del thread.
- `_shouldStop`: variable booleana que permanecerá como “true” mientras nadie haya solicitado un RequestStop().

```
public void DoWork(){

    Console.WriteLine("INICIO DEL THREAD");
    result = service.UploadFile(data2, strFile2, num);

    while (!_shouldStop){
        Console.WriteLine("*** Soy el thread y estoy activo hasta que me paren");
    }

    Console.WriteLine("FIN DE MI EJECUCION");
}
```

Para proceder a la creación de un hilo de esta clase previamente definida, es necesario pasar por parámetro a la constructora de la clase Thread esta función de inicio.

En el siguiente ejemplo hemos querido crear un *thread* de la clase MiClase, que contiene los métodos `DoWork()` y `RequestStop()` ya definidos:

```
MiClase obj = new MiClase();
Thread MiHilo = new Thread(obj.DoWork);
```

Para que el hilo MiHilo inicie su ejecución, debemos llamar a `MiHilo.Start()` desde el proceso padre. Esto iniciará la ejecución del código definido en la función `DoWork()`.

Para finalizar la ejecución, desde el proceso padre llamaremos a `MiHilo.Join()`, quedando éste suspendido hasta que se ejecute por completo el código de la región `RequestStop()`.

De este modo, en el desarrollo de esta aplicación, se ha invocado al método `UploadFile(...)` -encargado de enviar desde el cliente los ficheros de parámetros al servidor- en la región `DoWork()` definida en la clase genérica a la cual pertenecen todos los hilos del programa.

.NET Framework

¿Qué es .NET *Framework*?

.NET es el *framework* (conjunto de tecnologías) desarrollado por Microsoft y lanzado en el año 2002 orientado fundamentalmente a la creación de software para Internet. Proporciona una enorme variedad de librerías y ofrece una interconexión total entre todos los lenguajes de programación soportados por la suite Visual Studio. Además, abstrae a las aplicaciones de las diferentes plataformas de hardware utilizadas entre diferentes redes.

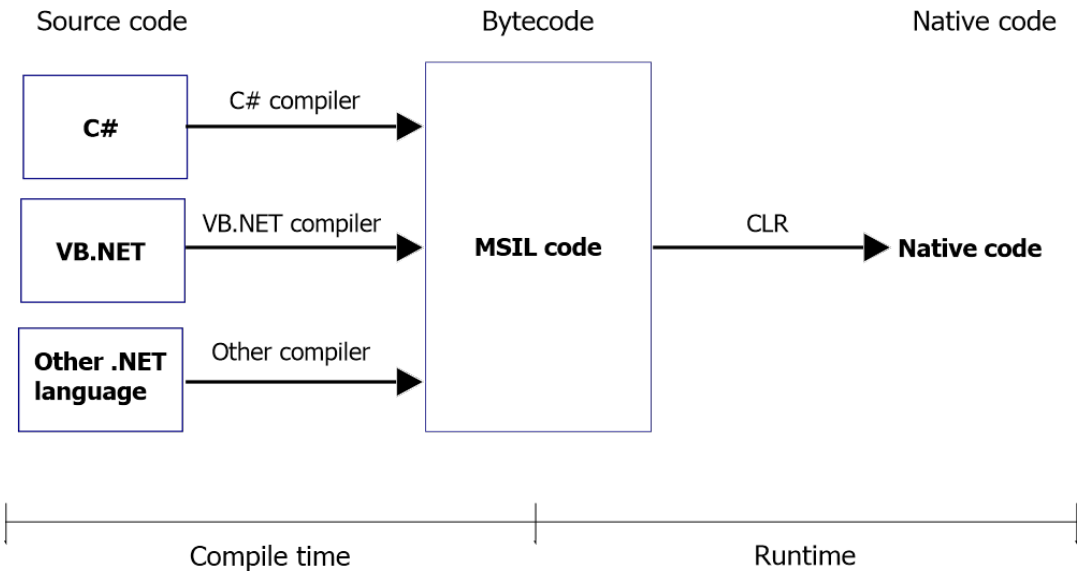
El objetivo básico de este conjunto de herramientas no es otro que el de obtener un entorno específicamente diseñado para el desarrollo y ejecución del software en forma de servicios, que puedan ser hechos públicos y acceder a ellos a través de Internet, de una forma independiente al hardware, software y lenguaje de programación utilizado en el desarrollo o el soporte de dicha aplicación. Este entorno global es lo que en Microsoft denominan Plataforma. NET, y los servicios ya mencionados son los que denominan como servicios web.

El esfuerzo invertido por Microsoft en el fácil y rápido desarrollo de aplicaciones orientadas a la web se tradujo en el diseño del nuevo lenguaje C# ya comentado en el apartado anterior, con el fin de utilizar éste como bandera del ágil desarrollo futuro en .NET .

Para el desarrollo y ejecución de aplicaciones en este nuevo entorno tecnológico, Microsoft proporciona el conjunto de herramientas conocido como .NET *Framework* SDK, que incluye compiladores de lenguajes como C#, Visual Basic.NET, Managed C++ y JScript.NET específicamente diseñados para crear aplicaciones para él.

Abstracción

El corazón del *framework* .NET es el CLR (*Common Language Runtime*). Se trata de la máquina virtual de este paquete, y es la responsable de administrar la ejecución de los programas de .NET . Además, realiza una compilación intermedia de todos los lenguajes soportados en un lenguaje de bajo nivel denominado CIL (*Common Intermediate Language*), que posteriormente se traduce a instrucciones máquina que son ejecutadas por la CPU. Ésta es la forma a través de la cual .NET consigue abstraer las aplicaciones de manera independiente al lenguaje de programación utilizado.



La última versión estable de .NET en el mercado es la 4.0, lanzada a la par que Visual Studio 2010 (Abril 2010). Como principal atractivo, esta versión cuenta con un soporte e integración total con Windows 7, y además está optimizada para el uso de múltiples procesadores y el enorme poder de procesamiento paralelo que el hardware más reciente posee, lo cual resultaba básico y esencial a la hora de poder llevar a cabo la parte más interesante de este proyecto: la explotación del paralelismo a gran escala.

Creación, desarrollo y despliegue de Servicios Web mediante Visual Studio e IIS

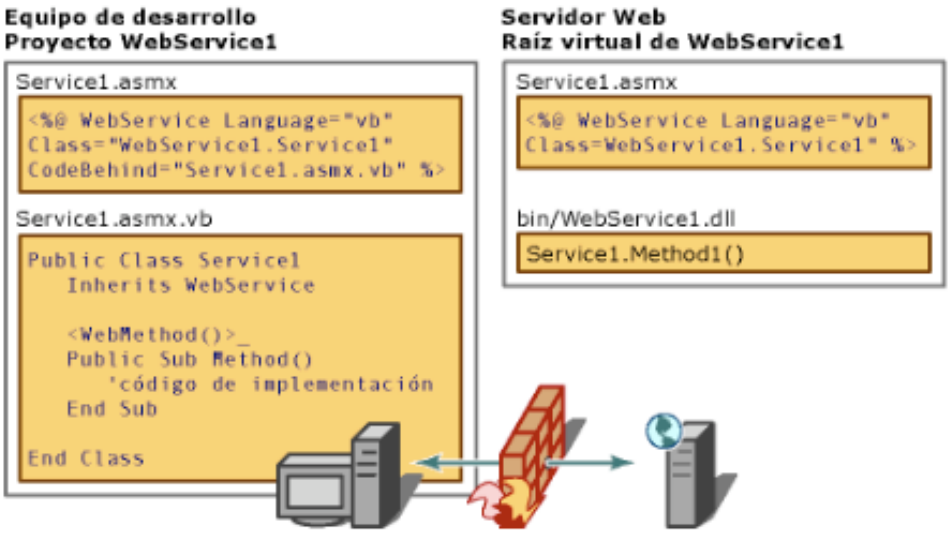
Para el desarrollo de este proyecto se han utilizado el *framework* de desarrollo Visual Studio y el servidor de aplicaciones IIS. Aunando fuerzas, las herramientas de Microsoft permiten un rápido y cómodo desarrollo y despliegue de servicios web, ahorrando al programador tiempo en labores de escritura de ficheros de configuración , apertura de puertos o preparación de servidores FTP, entre otros, y a la vez permitiéndole total libertad para adecuar estos parámetros en caso de considerarlo necesario.

Creación y desarrollo

Visual Studio ofrece al usuario la posibilidad de crear proyectos basados en distintos esquemas. Para el caso que nos ocupa, el sistema permite el uso de un proyecto plantilla de servicio web basado en el lenguaje C#. Los ficheros que compilan servicios web llevan la extensión ".asmx". Es un archivo de texto que sirve de punto de entrada direccionable del servicio Web XML. Hace referencia al código de ensamblados pre compilados, a un archivo de código subyacente o a un código contenido en el propio archivo .asmx. Veamos el ejemplo para este caso. El contenido del fichero es:

```
<%@ WebService Language="C#" CodeBehind="Service1.asmx.cs"
Class="WebService2.Service1"%>
```

El parámetro "Language" define el lenguaje en que será implementado el servicio, "*CodeBehind*" hace referencia al archivo que implementará el servicio con la extensión correspondiente a su lenguaje, y "*Class*" nos indica nombre de la clase que implementa el servicio. Sin embargo, mediante el asistente de Visual Studio prácticamente no nos tendremos que preocupar de estos detalles.



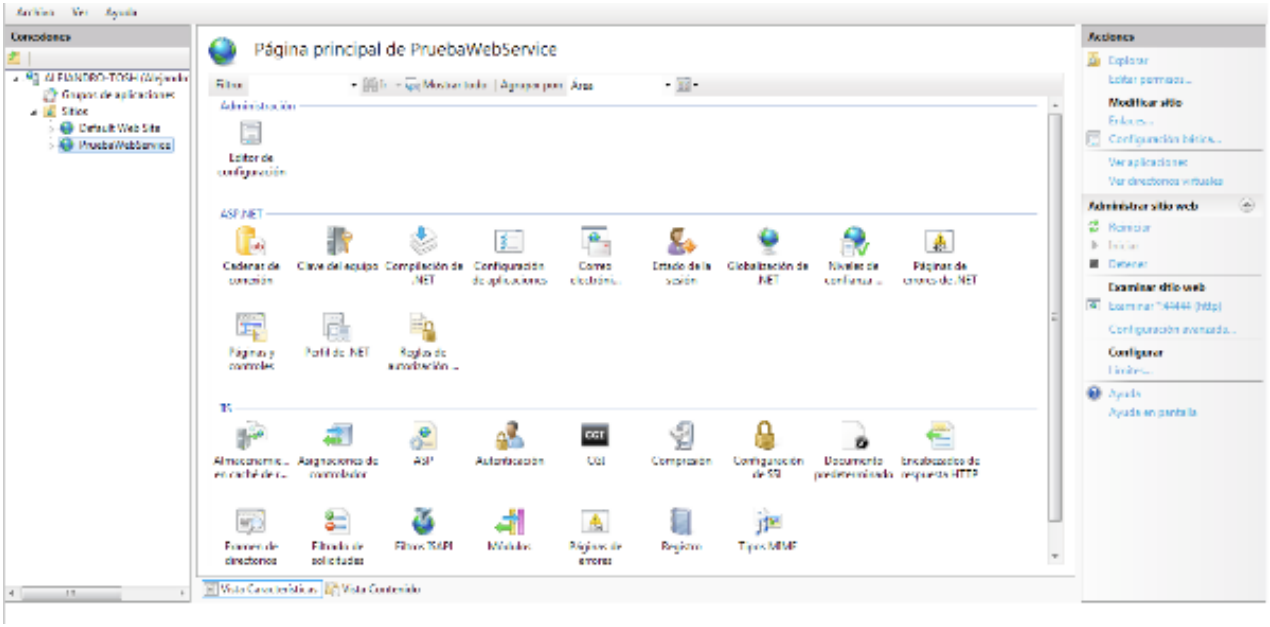
Como se ha podido observar, el fichero que implementa las funcionalidades será de tipo ".asmx." y una extensión correspondiente al lenguaje que implemente el servicio –"cs" para C#, "vb" para *Visual Basic*-.

Dentro de la clase que implementa el servicio, tendremos una constructora pública y los métodos que consideremos oportuno. De entre estos, tendremos que marcar con la anotación "*WebMethod*" aquellos métodos públicos que queramos hacer accesibles a aplicaciones que vayan a consumir el servicio. El atributo "*WebMethod*" proporciona algunas propiedades que permiten refinar el comportamiento del método, pero éstas quedan fuera del objeto de este estudio.

Despliegue

Una vez que hemos implementado nuestro servicio web, es hora de desplegarlo. Para el proceso de depuración durante el desarrollo, Visual Studio provee de un servidor local que nos permitirá hacer las pruebas oportunas. Para desplegar el servicio web y hacerlo accesible desde otros terminales, deberemos desplegarlo utilizando la herramienta Administrador de *Internet Information Services* -IIS-. Antiguamente esta herramienta tenía que ser adquirida o estaba incluida en determinadas compilaciones de Windows. Hoy en día viene integrada en Windows 7 y podremos acceder a ella a través de "Herramientas administrativas" en el Panel de Control.

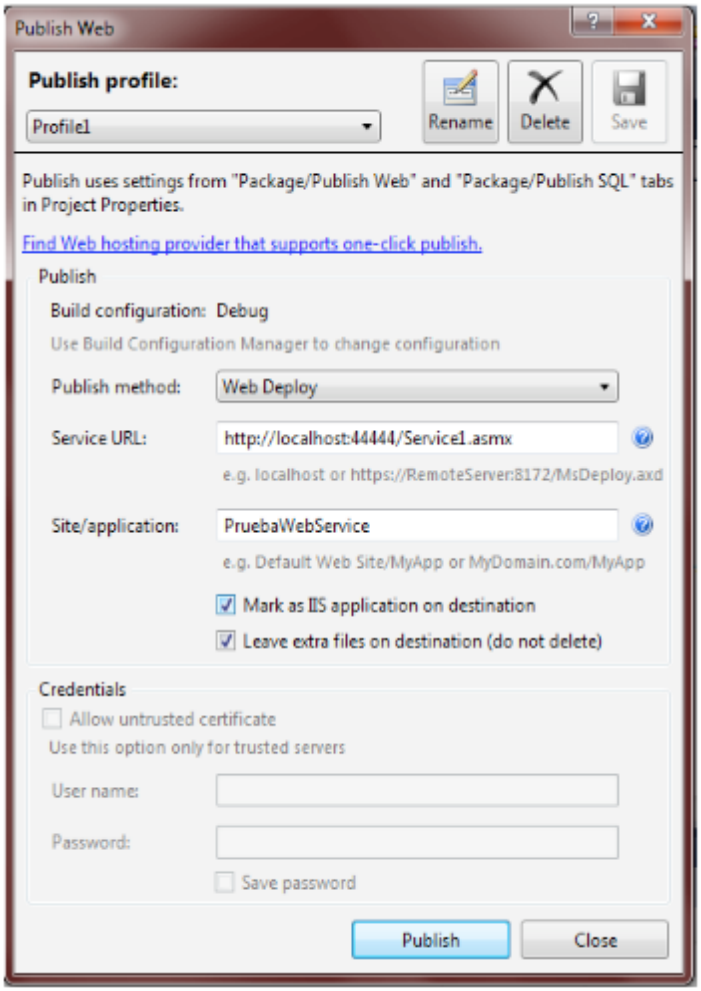
IIS: Es un servidor web que provee de servicios FTP, SMTP, HTTP/s y NNTP*. Se compone de varios módulos para procesar páginas ASP, ASP.NET y otras como las de PHP o Perl. Actualmente ostenta el puesto del tercer servidor más utilizado, con un doce por ciento del total de peticiones.



Administrador de IIS

Para poder desplegar el servicio que hemos creado en IIS, Visual Studio nos permitirá exportar el mismo mediante despliegue directo en el IIS, por FTP o creando una carpeta con los ficheros mínimos para ejecutarlo. Pinchando el botón derecho sobre nuestro proyecto y luego pulsando "*Publish*" podremos elegir las opciones de despliegue. De entre las opciones podremos elegir la dirección y puerto de despliegue. Si hacemos el despliegue mediante *Web Deploy*, tendremos que indicar la página que queremos que incluya el servicio. En caso de no existir en el servidor, IIS tomará automáticamente los pasos para crearla y desplegar el servicio. Si por el contrario elegimos la opción "*File System*", deberemos encargarnos nosotros mismos de crear la página y mapear la ruta del servicio en la máquina servidor mediante el administrador de IIS.

Cabe destacar que, ya que Visual Studio provee de un servidor interno para probar aplicaciones en desarrollo, si creamos un servicio web mediante el asistente, a la hora de desplegarlo deberemos borrar una serie de líneas de la sección "*configSections*" ya que IIS las incluye en sus propios ficheros de configuración y no se admitirá la duplicidad. Estas opciones se refieren al manejador de *scripts*, los servicios de autenticación y de rol, o el controlador de *Json*.



Capítulo 4: Conclusiones

Conclusiones Generales

A lo largo de este trabajo, se ha desarrollado como implementar un sistema que permita planificar una red logística de transporte de gas natural con el mínimo costo asociado al movimiento del mismo. El rasgo definitorio de este proyecto respecto a otros es que permite realizar análisis empíricos de sensibilidad sobre los parámetros del sistema valiéndose de computación en paralelo. Esto permite por un lado una gran flexibilidad a la hora de hacer la planificación debido a que se puede estudiar como afecta cada parámetro al sistema completo y por otro realizar este estudio en una fracción del tiempo que llevaría ejecutarlo de forma secuencial. El método de servicios web utilizado para implementar este paralelismo permite además poder desplegar la aplicación en computadores de todo tipo que posean capacidad de cálculo sobrante, no siendo necesario realizar una inversión en grandes servidores a menos que los problemas sean de una magnitud muy grande.

La planificación del sistema se realiza mediante la optimización de un modelo de programación lineal entera mixta (MILP) en el que se trata de minimizar el consumo de las estaciones de compresión a la vez que se satisface toda la demanda. Usando el resolutor matemático CPLEX, hemos realizado esta optimización.

La motivación de este proyecto reside en que realizar un análisis de este tipo manualmente resulta harto complicado, llegando a ser impracticable si se tarda más tiempo en realizar el cálculo que de el que tenemos para realizar la planificación. Por tanto, una automatización de este proceso como la que presentamos permite optimizar los recursos del sistema, repercutiendo en el coste de operación del mismo.

Resultados

Tras probar la aplicación en diferentes máquinas y con modelos de tamaños diferentes, hemos obtenido que la proporción de tiempo ahorrado al realizar el análisis de sensibilidad paralelamente es mayor cuanto más complejo es de resolver el modelo. Esto se traduce en que para modelos muy grandes o que se pretendan planificar para un número elevado de periodos, la aplicación consigue alcanzar la solución óptima en una fracción del tiempo que tardaría en hacerlo secuencialmente. Para modelos pequeños (Como el modelo de prueba presentado en el capítulo anterior) la cantidad de tiempo que se pierde en realizar la conexión a través de la red hace que no compense utilizar varias computadoras, a no ser que el análisis de sensibilidad incluya un rango muy amplio de valores.

En cualquier caso, las planificaciones reales que se realizarían en una aplicación como esta se encuentran al menos dentro de uno de ambos escenarios, por lo que los beneficios potenciales para una empresa que se valga de esta herramienta serán evidentes. No debemos de olvidar que, al tratarse de hacer una planificación, el tiempo es un factor vital. Una solución, por muy buena que sea, será inútil si para el momento que se obtiene ya no se puede aplicar. Los márgenes con los que se opera en este caso son estrechos y cualquier ahorro de tiempo que se pueda obtener significa aumentar los beneficios.

Para servidores con la misma capacidad de cómputo y carga de trabajo, se consiguen efectivamente ejecuciones muy próximas a 1/n del tiempo que tardaría en hacerse secuencialmente. Hemos creado modelos relativamente complejos que se mueven en el rango de las decenas de minutos, por tanto los beneficios de la aplicación son aparentes.

El único factor que puede hacer que este rendimiento se degrade es que los computadores sean muy dispares en cuanto a potencia o carga de trabajo se refiere. Como la solución óptima no queda compuesta hasta que se obtiene la respuesta de cada uno de ellos, el hecho de que uno se retrase implica que todo el sistema lo haga también. Para evitar esto, se utiliza el procedimiento que se encarga de descubrir que servicios web están activos para realizar una prueba de rendimiento de cada uno. En base a esto se puede planificar la cantidad de mensajes que se enviarán a cada servidor para equilibrar las cargas de trabajo.

Posibles Ampliaciones

Se ha comentado que esta aplicación utiliza una de las clases de computación en la nube a la hora de implementar el paralelismo, concretamente el de la nube privada. Un paso más allá sería colocar la aplicación en un servidor de la nube pública del tipo Plataforma como Servicio (PaaS) como podría ser un servidor de Amazon y de esta forma dejar la gestión de la infraestructura a una empresa externa, minimizando así los costes. Esto plantearía dos inconvenientes: Por un lado la sensibilidad de la información que se dejaría vagar por la red, lo cual podría protegerse con los adecuados algoritmos de encriptación y por otro la política de IBM respecto a colocar CPLEX en uno de estos servidores es restrictiva. Como alternativa a CPLEX si se desease llevar a cabo esta tarea,

se ha investigado sobre el uso de *Gurobi*, otro resolutor matemático que también trabaja con OPL y dispone de licencias específicamente diseñadas para esta tarea. *Gurobi* incluso ofrece tutoriales para migrar desde CPLEX, por tanto podría realizarse con relativamente poco esfuerzo.

Anexo: Licencias

IBM ofrece diferentes tipos de licencias para CPLEX dependiendo de el uso que se quiera dar al mismo. Esto incluye desde versiones gratuitas de prueba que están bastante limitadas en cuanto a capacidad de cálculo y que no permiten ser llamadas externamente, pasando por licencias académicas como la que hemos utilizado hasta llegar a versiones completas. El costo de cada una de ellas es a menudo confuso y es necesario ponerse en contacto con IBM para obtener un precio dependiendo de en que campos vaya a usarse el resolutor.

Anexo: Cesión de Derechos

Adrián García Romero, Miguel Mena Jiménez y Alejandro Soto Rebollo, los autores del proyecto y abajo firmantes autorizamos a la Universidad Complutense a difundir y utilizar con fines académicos, no comerciales y mencionando expresamente a sus autores, tanto la propia memoria, como el código, la documentación y/o el prototipo desarrollado.

Agradecimientos

A nuestro director de proyecto, José Jaime Ruz Ortiz, por su paciencia y consejo.

A mis padres y a Raquel por su apoyo incondicional.

A mi familia por su apoyo continuo, a Yasmine por estar siempre ahí.

Bibliografía

Gorostegui López-Alonso, "Modelado, Optimización y Planificación de una Red de Distribución de Gas Natural", Trabajo Fin de Máster (Universidad Complutense de Madrid / Universidad Nacional de Educación a Distancia) 2011

Mokhtar S. Bazaraa, John J. Jarvis, Hanif D. Sherali, "Programación lineal y flujo en redes", Limusa, 2004

Ramos Méndez, Eduardo, "Programación Lineal y Métodos de Optimización" Uned, 1991

A. H. Land and A. G. Doig, "An automatic method of solving discrete programming problems". Econometrica : pp. 497–520, 1960

Cornuejols, Gerard . "Revival of the Gomory Cuts in the 1990s. Annals of Operations Research": pp. 63-66, 2007

Herrán González, A., “Modelado, Planificación y Control de Sistemas de Distribución de Gas y Derivados del Petróleo”, Tesis Doctoral (Universidad Complutense de Madrid) 2008.

Moyer, Christopher M. "Building applications in the cloud: concepts, patterns, and projects", Addison-Wesley, 2011