Universidad Complutense de Madrid

Facultad de Informática

Implementación de un simulador de rendimiento, consumo y coste para Centros de Datos integrados en un Smart Grid

Alumnos **Fernando Baraibar López**
Roberto Montero Cobo de Guzmán

Directores de proyecto **José Luis Ayala Rodrigo**
Marina Zapater Sancho

Trabajo de Fin de Grado

Ingeniería de Computadores

Junio de 2017

Índice general

Palabras clave

Palabras clave en Español

- Modelado y simulación
- Sistemas de eventos discretos
- Formalismo DEVS
- Sistemas de ventilación manual mecánica
- Centro de Proceso de Datos
- Eficiencia energética de centros de proceso de datos

Keywords in English

- Modeling and simulation
- Discrete events system
- DEVS formalism
- Freecooling
- Data Center
- Energy Efficiency in Data Centers

Resumen

Resumen en Español

Actualmente uno de los mayores costes de un Centro de Proceso de Datos es el consumo eléctrico, especialmente el generado para el funcionamiento de los sistemas de refrigeración.

Los simuladores constituyen una herramienta imprescindible para optimizar su funcionamiento de la forma más eficiente posible.

A su vez, los simuladores deben ejecutarse en tiempos razonables, además de permitir su ejecución con distintas configuraciones de los sistemas cuyo comportamiento se quiere simular.

Este proyecto consiste en la elaboración y desarrollo de un modelo de consumo de un Data Center utilizando las ecuaciones descritas en los artículos *"Dynamic Workload and Cooling Management in High-Efficiency Data Centers"* y *"Leakage-Aware Cooling Management for Improving Server Energy Efficiency"*, reseñados en la bibliografía [1] y [2], empleadas anteriormente en el simulador DC-Sim, generando una infraestructura base para futuros desarrollos.

El nuevo modelo, basado en el formalismo DEVS, es fácilmente escalable y permite la ampliación con nuevas funcionalidades, además de permitir la implementación de cualquier topología y tamaño de Data Center, con diferentes cargas de trabajo.

Summary in English

Currently, one of the highest costs of a Data Processing Center comes from electrical consumption, especially the one produced to allow proper functioning of the cooling systems.

Simulators are an essential tool to optimize the performance of the cooling systems in the most efficient way possible.

Thus, simulators must be executed within reasonable times, and their execution must be compatible with different settings of those systems whose behavior is to be simulated.

The present project deals with the creation and development of a consumption model for a Data Center using the equations described in the following articles: "Dynamic Workload and Cooling Management in High-Efficiency Data Centers.ªnd "Leakage-Aware Cooling Management for Improving Server Energy Efficiency ", referenced in the bibliography, [1] and [2]. These equations were previously used in the DC-Sim simulator, generating a basic infrastructure for future developments.

The new model, based on the DEVS formalism, is easily scalable and allows expansion with new functionalities, besides allowing implementation of any topology and size of a Data Center,

with different workloads.

Capítulo 1

Introducción

La rápida evolución de la tecnología ha permitido un desarrollo de forma exponencial de los sistemas de información. La posibilidad de instalar varios equipos más potentes en el espacio que hace unos años ocupaba uno sólo ha generado un considerable incremento en la potencia necesaria para el funcionamiento del conjunto.

El aumento de la densidad de equipos en las instalaciones hace que se genere más calor y de forma más concentrada, por lo que ha sido necesario desarrollar sistemas de refrigeración más sofisticados y eficientes, con el fin de evitar sobrecargas térmicas. Es por todo esto que los Data Center cada vez consumen más energía.

Mientras que el coste de los equipos se ha reducido, el de la electricidad y los sistemas de refrigeración se ha incrementado considerablemente, dando lugar al extremo de que es fácil que llegue a resultar más caro mantener en funcionamiento un equipo que el propio equipo.

Estas circunstancias hacen imprescindible un profundo estudio a la hora de afrontar la instalación de un Data Center. El coste energético puede variar considerablemente dependiendo, por ejemplo, de su localización geográfica, en cuanto que la temperatura exterior influirá en el gasto asociado al proceso de refrigeración. También su topología, las diferentes cargas de trabajo o los distintos tipos de servidores instalados afectará directamente sobre la factura eléctrica.

Para poder tomar decisiones sobre la ubicación y características de un Data Center resulta de extrema utilidad la utilización de sistemas que simulen su comportamiento con las diferentes condiciones que puedan afectar a su consumo y eficiencia.

Las simulaciones permiten abstraer la complejidad de un sistema para generar una herramienta que permita comprender con mayor facilidad la evolución de éste en el tiempo.

Introduction

The fast evolution of technology has allowed an exponential development of information systems. The possibility to install several and more powerful machines in the same space which, a few years ago, was taken up by just one equipment, has caused a considerable increase in the power needed to operate the whole set.

Increasing the equipment's density in the facilities causes more heat, and enhances its concentration. Therefore, it has been necessary to develop more sophisticated and efficient cooling

systems, in order to avoid thermal overloads. This is the reason why Data Centers consume more and more energy.

While equipment costs have been reduced, the cost for electricity and cooling systems has increased considerably. As a result, the act of keeping a computer functioning is likely to be more expensive that the computer itself.

These circumstances create the need for a thorough study when facing the installation of a Data Center. The energy cost may vary substantially depending, for instance, on the geographic location, since outside temperature will influence the cost related to the cooling process. Other factors such as topology, different workloads or different types of installed servers will directly affect the electricity bill.

To facilitate the decision-making process with respect to the location and characteristics of a Data Center, it is extremely useful to employ systems to simulate its behavior with the different conditions that might affect its consumption and efficiency.

Simulations facilitate the abstraction of a system's complexity in order to generate a tool that permits a better understanding of its evolution in time.

1.1. Antecedentes

La contribución de los Centros de Proceso de Datos al consumo eléctrico en Europa se está incrementando sensiblemente en los últimos años. Los vertiginosos avances técnicos en materia de hardware han conseguido que el incremento de consumo de energía de los nuevos equipos, mucho más potentes y eficientes, sólo se haya incrementado entre un 5 % y un 8 %, pero la gran cantidad de servidores que pueden llegar a conformar un Data Center hace que el consumo global alcance unos valores muy elevados [6].

Para mejorar la eficiencia global es imprescindible el desarrollo de modelos de Data Center que optimicen su consumo y rendimiento. Pero no se pueden adoptar políticas o algoritmos sin haberlas evaluado previamente, para lo cual la simulación se presenta como un instrumento muy importante para poder evaluar el comportamiento de un Data Center en cuanto al consumo, rendimiento, costes, etc.

Un simulador debe permitir incorporar o modificar nuevos equipos a la configuración del Data Center, por lo que debe estar basado en plugins con la posibilidad de evolucionar al tiempo que se introducen nuevos equipos informáticos o técnicas de refrigeración. Los simuladores actuales no cumplen estos requisitos, ya que la simulación se ha centrado en predecir la termodinámica de los Data Center en tiempo de diseño utilizando software de Dinámica Computacional de Fluidos (CFD)(Singh, Singh, Parvez y Sivasubramaniam 2010), cuya adaptación a diferentes configuraciones es excesivamente complejo, implicando un elevado coste.

Se han desarrollado múltiples simuladores, pero que sólo abordan la dinámica térmica o son aplicaciones ad hoc.

SimWare y **CoolSim** se centran en la sala de datos, sin tener en cuenta el rendimiento, la planificación, la asignación de tareas y los costes operativos.

CloudSim se centra específicamente en escenarios de Cloud Computing.

BSC Slurm Simulator y **SST** se centran en el rendimiento obtenido a través de la programación y asignación de tareas HPC.

iCanCloud proporciona herramientas de configuración relativamente sencillas, algo poco común en el resto de simuladores.

CloudNet-Sim ++ ha mejorado la usabilidad y permite la creación de redes. Tanto este simulador como **CloudSim** presentan un motor de simulación completamente acoplado con el modelo, lo que hace muy difícil la incorporación de nuevos modelos y funcionalidades.

1.2. Objetivo

Los simuladores y modelos actuales son sistemas muy acoplados, de manera que cualquier modificación en el modelo implica a la plataforma de simulación. El objetivo del presente trabajo es el desarrollo e implementación de un modelo que permita la abstracción de la estructura y comportamiento de un data center utilizando el formalismo DEVS, de manera que su comportamiento coincida con el simulado por DC-Sim.

La ventaja de utilizar DEVS es que existen simuladores, como xDEVS, para la ejecución de los modelos elaborados con este formalismo. Cualquier modificación en el modelo, como la inclusión o sustitución de componentes, no repercute en el simulador, siendo además posible la inclusión de nuevos elementos de monitorización.

El simulador **DC-Sim** se desarrolló para la simulación del consumo del MGHPCC (Massachussetts Green High Performance Computing Center) [1]. A partir de su implementación hemos desarrollado el modelo **SFIDE** (Simulation Framework and Infrastructure for Data cEnters), basado en el formalismo DEVS. Para ello ha sido necesario entender el concepto y funcionamiento del sistema de refrigeración de un Data Center.

El estudio y análisis del código del simulador DC-Sim nos ha permitido estudiar su comportamiento y poderlo trasladar a la nueva implementación.

Teniendo como referencia los datos reales obtenidos mediante DC-Sim, la validación del modelo **SFIDE** se ha llevado a cabo mediante la ejecución de diferentes escenarios en ambos sistemas y la comprobación de la similitud de los datos obtenidos.

La flexibilidad y escalabilidad del nuevo modelo permitirá la configuración de distintos Data Center de una manera sencilla y eficiente.

1.3. Plan de trabajo

La elaboración de este trabajo se ha llevado a cabo siguiendo una serie de fases, realizando reuniones periódicas con el director de proyecto con el fin de validar el trabajo realizado.

Fase 1. Estudio y comprensión del formalismo DEVS. Asistimos a clases del profesor Dr. José Luis Risco, que nos explica al especificación de los modelos DEVS atómico y acoplado, facilitándonos un manual que estudiamos.

Fase 2. Instalación y puesta a punto del simulador DC-Sim. Nos facilitan la implementación del simulador en C++, procediendo a su instalación. Para esta tarea recibimos colaboración de Ignacio Penas, estudiante de Máster de la UPM, que nos tuteló en la instalación de las bibliotecas necesarias y la configuración del programa.

Fase 3. Estudio y análisis del simulador DC-Sim. Una vez instalado, analizamos las distintas clases implementadas para entender su comportamiento y poder trasladarlo al nuevo modelo.

Fase 4. Diseño de la arquitectura del nuevo modelo. A partir de un boceto facilitado por la directora de proyecto establecemos la arquitectura, que sufre numerosas modificaciones cuando se comienza el diseño del modelo SFIDE. Después de analizar diversas opciones, se fija la estructura definitiva, definiendo los componentes que conformarán el modelo.

Fase 5. Diseño y programación del modelo SFIDE. A partir de la estructura establecida, comenzamos la programación del comportamiento de los distintos componentes en C++. Comprobamos funcionamiento ejecutándolo con un escenario de prueba, realizando las modificaciones pertinentes.

Fase 6. Ejecución de SFIDE y DC-Sim con diversos escenarios. Nos facilitan dos escenarios de distintas dimensiones para ejecutarlos con ambos modelos, con el fin de comparar los respectivos resultados y poder validar la nueva implementación. Desarrollamos los scripts necesarios para adaptar los escenarios para ser utilizados por SFIDE.

Fase 7. Resolución de errores del modelo a partir de los nuevos escenarios. Una vez ejecutados los nuevos escenarios, se detectan numerosos "bugs"mediante el uso de scripts diseñados para analizar los resultados, que son subsanados.

Fase 8. Comparación de resultados. Los resultados de ambas ejecuciones son analizados y comparados mediante los scripts pertinentes, permitiendo validar los resultados del modelo SFIDE.

Fase 9. Estudio de LaTeX. Estudiamos el sistema de composición de textos LaTeX para la elaboración del presente documento.

Fase 10. Redacción de la Memoria.

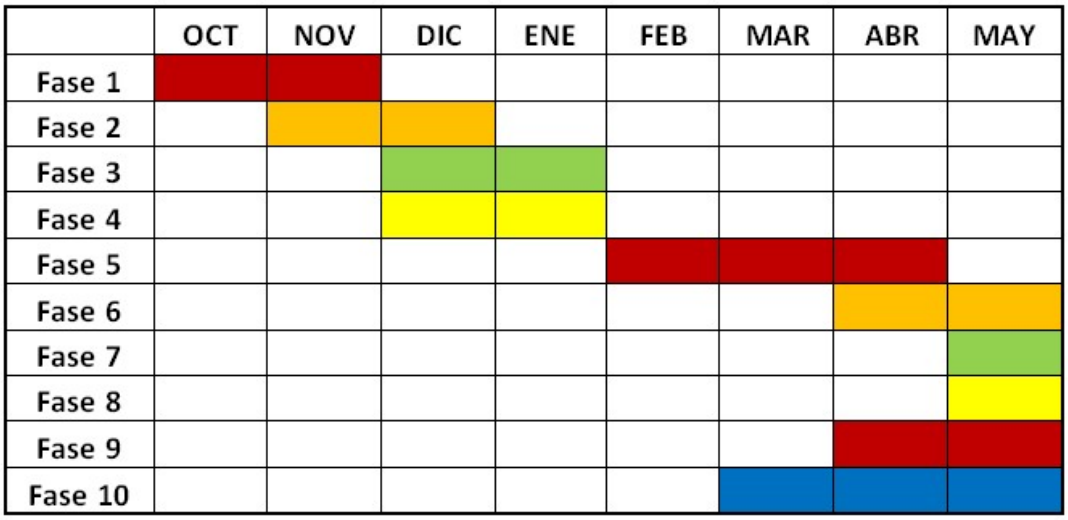

Figura 1.3.1: Diagrama de Gantt. Fases de desarrollo

En la figura 1.3.1 se muestran los tiempos de cada una de las fases de este proyecto de fin de grado. Ninguna de las tareas que se realizaron fue mutuamente excluyentes y procuramos que los dos miembros del equipo trabajásemos en todas y cada una de las tareas referentes a cada fase.

Capítulo 2

Visión general

2.1. El formalismo DEVS

DEVS, acrónimo de Discrete Event System Specification, es un formalismo modular y jerárquico creado por Bernard P. Ziegler en los años 70 para el modelado y simulación de sistemas de eventos discretos, donde las entradas, salidas y estados son constantes por intervalos [4]. Estos intervalos de tiempo entre ocurrencias son variables, aportando ventajas frente a otros formalismos con intervalos únicos.

Un modelo DEVS consta de una base de modelos básicos, llamados atómicos, que se combinan para formar modelos acoplados, junto con un conjunto de relaciones que indican cómo están conectados entre ellos, formando una estructura jerárquica.

La primera descripción del formalismo DEVS recibe el nombre de DEVS *clásico*, ya que en 1.996 *Chow* y *Ziegler* realizaron una revisión del mismo con el fin de explotar las posibilidades de computación paralela. Esta revisión dio paso al formalismo DEVS *paralelo* (P-DEVS), que implicaba la supresión de algunas restricciones del DEVS *clásico* [7].

El formalismo DEVS *paralelo* difiere del DEVS *clásico* en que aquél permite la activación simultánea de eventos en diferentes componentes del modelo, y la generación igualmente de forma simultánea de las correspondientes salidas. También permite la llegada simultánea de múltiples eventos a un mismo puerto de entrada y la generación de varios eventos en un mismo puerto de salida.

En el presente trabajo se utilizará el formalismo DEVS *paralelo*.

2.1.1. Modelo Atómico

El modelo atómico DEVS *paralelo* incluye la siguiente configuración:

- Un conjunto de puertos de entrada, por donde se reciben los eventos de entrada al sistema.
- Un conjunto de puertos de salida, por donde se envían los eventos producidos en el modelo.
- Un conjunto de estados.
- Una función de avance de tiempo, que determinará la duración del estado actual.

- Una función de transición interna, que determina el estado al que debe pasar el sistema al finalizar el tiempo determinado por la función de avance de tiempo.
- Una función de transición externa, que determina el cambio a otro estado cuando se recibe una entrada. Es función del estado actual, del puerto por el que se recibe la entrada, el valor recibido y el tiempo transcurrido en el estado actual.
- Una función de transición confluyente, que permite especificar cuál es el nuevo estado cuando se produce la confluencia de una transición interna y una externa.
- Una función de salida, que genera una salida inmediatamente antes de producirse una transición interna.

Un modelo DEVS atómico se especifica como:

$$M = \langle X_M, S, Y_M, \delta_{int}, \delta_{ext}, \delta_{con}, \lambda, ta \rangle \tag{2.1}$$

donde:

- X_M : Conjunto de eventos de entrada, formados por pares puerto-valor:

$$X_M = \{(p, v) | p \in InPorts, v \in X_p\} \tag{2.2}$$

- S : Conjunto de estados secuenciales.
- Y_M : Conjunto de eventos de salida, formado por pares puerto-valor:

$$Y_M = \{(p, v) | p \in OutPorts, v \in Y_p\} \tag{2.3}$$

- $\delta_{int} : S \to S$, función de transición interna.
- $\delta_{ext} : Q \times X_M^b \to S$, función de transición externa, con

$$Q = \{(s, e) | s \in S, e \in [0, ta(s)]\} \tag{2.4}$$

  Cada elemento del conjunto *Q* está formado por dos valores: *(s,e)*, donde *s* es un elemento del conjunto *S* y *e* es un número real positivo: el tiempo transcurrido desde la anterior transición del estado.

- $\delta_{con} : Q \times X_M^b \to S$, función de transición confluyente
- $\lambda : S \to Y^b$, función de salida que dependerá de la duración del estado.
- $ta : S \to \mathbb{R}_0^+$,función de avance de tiempo.

Al definir la función de transición externa, X_M^b representa una *"bolsa"* de parejas *(puerto,evento)*, que contempla el hecho de que pueden llegar múltiples eventos simultáneamente a un mismo puerto, o a diferentes puertos.

De igual manera, la función de salida λ genera una *"bolsa"* de eventos de salida, Y^b.

La función de transición confluyente, δ_{con}, tiene como finalidad definir un nuevo estado cuando en un mismo instante se recibe una *"bolsa"* de eventos de entrada y está planificada una transición interna de estado. Esta función permite determinar cuál es el nuevo estado cuando se produce la confluencia de una transición interna y otra externa.

En cualquier momento, el sistema se encuentra en un estado s. Si no se produce ningún evento de entrada, el sistema permanecerá en dicho estado s durante un tiempo $ta(s)$. Si $ta(s) = 0$, el estado se considera de transición, en el que no es posible que suceda ningún evento externo. Si, por el contrario, $ta(s) = \infty$, el sistema permanecerá en tal estado, denominado pasivo, hasta que reciba algún evento externo.

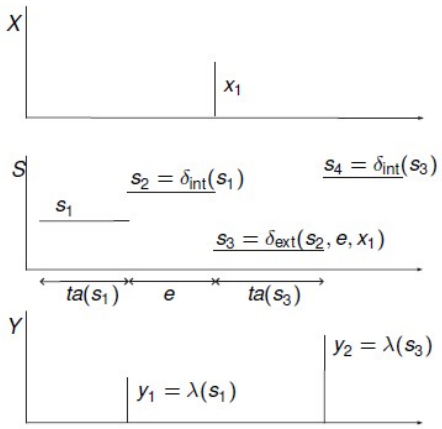

Figura 2.1.1.a: Diagrama de tiempo del modelo atómico

Cuando el tiempo transcurrido en un estado S supera a $ta(s)$, el sistema genera un evento de salida $\lambda(s)$, pasando al estado $s' = \delta_{int}(s)$, el estado de transición interna y sabemos que no se ha producido ningún evento externo durante el tiempo permanecido en el estado s.

Si antes de producirse la transición interna ocurriera un evento externo con valor x, mientras el sistema se encuentra en el estado (s, e), siendo $e \leq ta(s)$, el sistema pasaría al estado $\delta_{ext}(s, e, x)$, pero sin producirse ningún evento de salida.

2.1.2. Modelo Acoplado

Un modelo acoplado está constituido por un conjunto de modelos atómicos y/o acoplados, cuyas entradas y salidas se encuentran interconectadas entre ellos y con el exterior del sistema. Un modelo acoplado N se define:

$$N = \langle X, Y, D, \{M_d | d \in D\}, EIC, EOC, IC \rangle \tag{2.5}$$

donde:

- X : Conjunto de eventos de entrada.

- D : Conjunto de los nombres de los componentes (atómicos o acoplados).
- Y : Conjunto de eventos de salida.
- M_d : modelo DEVS para cada $d \in D$.
- EIC : conjunto de conexiones de entrada externas.
- EOC : conjunto de conexiones de salida externas
- IC : conjunto de conexiones internas.

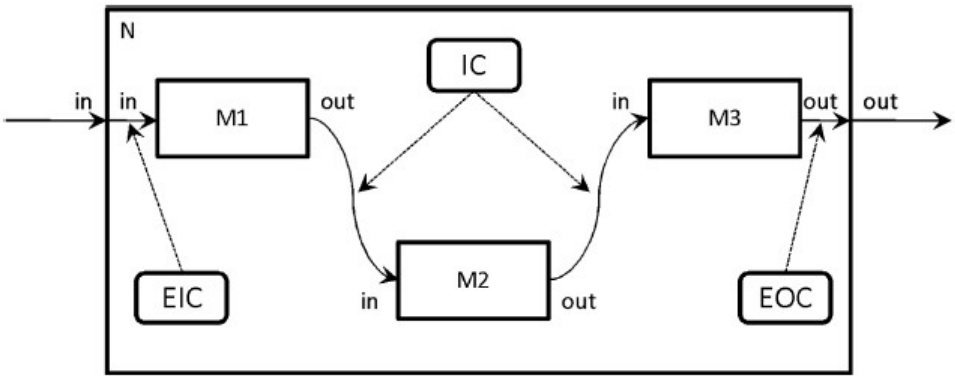

Figura 2.1.2.a: Modelo acoplado DEVS

En el ejemplo se muestra el modelo acoplado N, con tres componentes M1, M2 y M3, que pueden ser a su vez atómicos o acoplados, y sus interconexiones mediante los correspondientes puertos de entrada y salida, así como las conexiones externas de entrada (EIC) y de salida (EOC).

La definición formal del modelo representado en la figura 2.1.2.a sería:

$$N = \langle X, Y, D, \{M_d | d \in D\}, EIC, EOC, IC \rangle \tag{2.6}$$

donde:

- X = Conjunto de eventos de entrada.
- Y = Conjunto de eventos de salida.
- $D = \{M_1, M_2, M_3\}$.
- $M_d = \{M_{M1}, M_{M2}, M_{M3}\}$.
- $EIC = \{(N, in) \rightarrow (M_1, in)\}$.
- $EOC = \{(M_3, out) \rightarrow (N, out)\}$.
- $IC = \{(M_1, out) \rightarrow (M_2, in), (M_2, out) \rightarrow (M_3, in)\}$.

2.1.3. Simulación de modelos DEVS

La formalización DEVS permite la simulación de modelos muy complejos de una manera sencilla y eficiente.

La estructura del simulador se obtiene a partir del modelo definido, de la siguiente forma:

- A cada modelo atómico DEVS se le asocia un algoritmo denominado *Simulator*, que realiza la simulación del modelo DEVS atómico.
- A cada modelo acoplado DEVS se le asocia un algoritmo denominado *Coordinator*, que realiza la simulación del modelo DEVS acoplado.
- En la raíz de la jerarquía del simulador se sitúa el algoritmo *Root Coordinator*.

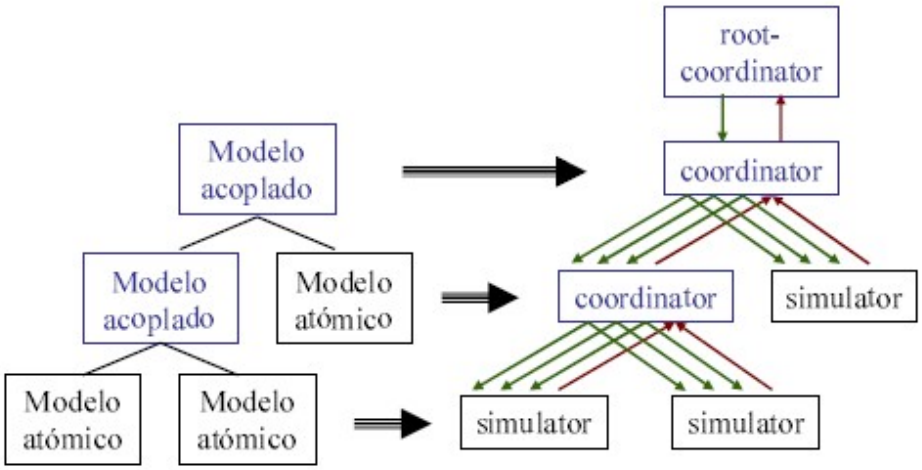

Figura 2.1.3.a: Estructura del simulador DEVS.

El algoritmo de simulación es el siguiente:

1. El *Coordinator* solicita que cada *Simulator* facilite la hora de su siguiente evento, seleccionando el que presente la menor.

2. Se actualiza el tiempo de simulación *t* hasta el tiempo de transición de su δ_{int}.

3. Se ejecutan las funciones λ y δ_{int} correspondientes al componente atómico asociado al *Simulator* seleccionado.

4. Tras la ejecución de la función λ se propagan las salidas generadas hacia los elementos conectados, ejecutándose las funciones δ_{ext} de éstos.

5. Se vuelve a ejecutar el paso 1.

2.2. Construcción de modelos de data center

2.2.1. Introducción

Un centro de procesamiento de datos es una instalación formada por múltiples servidores cuyo objetivo es el procesamiento, almacenamiento y acceso de datos. La propia definición de data center abarca tanto a los pequeños armarios de una oficina como a las grandes granjas de servidores de empresas como Google.

2.2.2. Componentes de un data center

Los datacenters están conformados por un sistema de refrigeración, un sistema distribuido de servidores y, opcionalmente, un sistema de alimentación auxiliar.

- Sistemas de refrigeración: Un datacenter, al estar constituido por servidores, genera grandes cantidades de calor que, con los medios integrados de disipación de calor no son capaces de extraerlo, acumulándose en la estancia del CPD y afectando negativamente al rendimiento de las CPUs. Por esta razón se requiere un sistema adjunto que sea externo a los servidores y que retire el aire caliente que dichos servidores generan. Existen 2 tipos de sistemas de refrigeración externos a los servidores:

  - Refrigeración In-Rack: Se trata de un sistema de refrigeración *"close-coupled"*, que acerca la fuente fría (el agua) a la fuente de calor (los servidores), que es altamente eficiente en términos de consumo energético. Para cada rack o agrupación de racks se le asigna una toma de entrada de agua fría, que se calentará a partir del aire caliente generado por los servidores del Rack, dando lugar a agua caliente que se irá de vuelta por el mismo circuito, como se muestra en la figura 2.2.2.a.

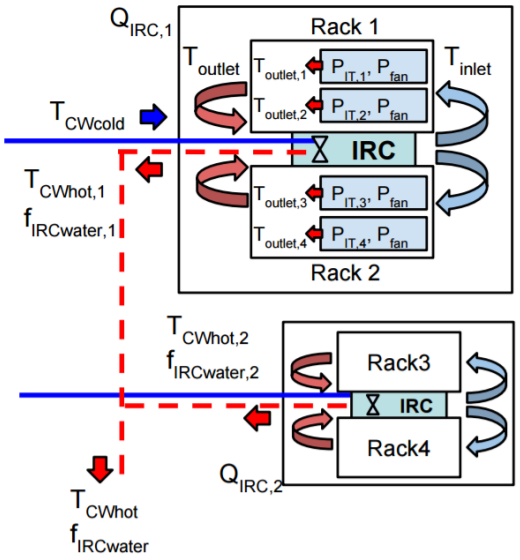

Figura 2.2.2.a:Sistema de refrigeración In-Rack[1]

Los sistemas de refrigeración In-Rack requieren de 3 componentes externos a los Rack.

1. Bomba de agua: Se encarga de mover el agua caliente y el agua fría a través del circuito de refrigeración.
2. Enfriador: El enfriador se encarga de enfriar el agua del circuito siempre que el intercambiador de calor no sea suficiente para enfriar el agua a la temperatura preestablecida recibida de la refrigeración de los servidores.
3. Intercambiador de calor: Un intercambiador de calor es un sistema que permite que dos fluidos intercambien calor sin necesidad de mezclarse. Está formado por una cámara que contiene uno de los fluidos y una bobina dentro de la cámara con el fluido a mayor temperatura. En el MGHPCC el fluido frío (a la temperatura del exterior) viene de la torre de refrigeración y el agua caliente viene de la refrigeración de la sala de servidores.
4. Torre de refrigeración. La función de la torre de refrigeración es enfriar el agua procedente del intercambiador de calor, enviándola de vuelta a éste.

- Refrigeración mediante CRAC (Computer Room Air Conditioning): A diferencia de la refrigeración In-Rack, no se basa en la aproximación del elemento refrigerante a la fuente de calor, sino que dispone de un intercambiador de calor a través del cual circula el agua fría procedente de los componentes externos (bomba, enfriador, intercambiador de calor y torre de refrigeración), que absorbe el calor generado por los servidores, enfriando el aire y haciéndolo circular de vuelta a la zona en que se encuentran los racks. Este sistema es mucho menos eficiente que la refrigeración In-Rack.

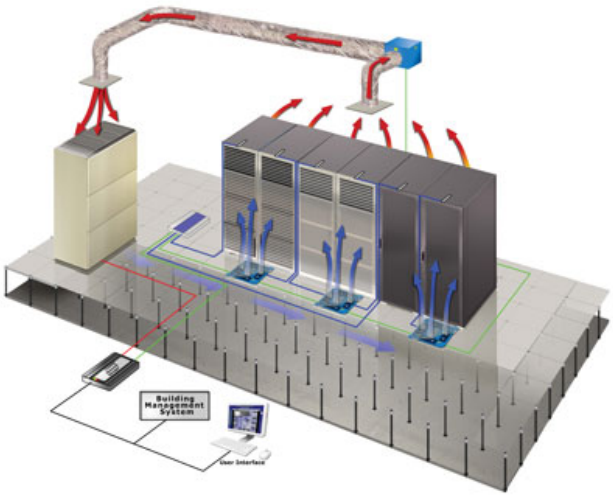

Figura 2.2.2.b. Refrigeración mediante CRAC

- Sistemas de TIC: Se encargan de realizar el trabajo de cómputo del CPD.
- Sistemas de alimentación: Un CPD debe estar disponible en torno al 99 % del tiempo, por

ello exigen fuentes de alimentación auxiliares, para que en caso de que falle la Red eléctrica el CPD no se apague. Suelen estar conformados por baterías y/o generadores eléctricos de diesel o gasolina.

2.2.3. DC-Simulator

DC-Sim nace como un proyecto para la simulación del consumo del MGHPCC (Massachusetts green high performance computing center) con el objetivo de ser una herramienta para la investigación en técnicas de reducción del consumo en los centros de procesado de datos [1].

La finalidad de DC-Sim es conseguir los datos de consumo de un centro de procesado de datos a partir de la topología de dicho CPD, una serie de tareas y los valores de la temperatura ambiental. Existen dos posibilidades de funcionamiento en DC-Sim:

- Funcionamiento estático: Como su propio nombre indica, el funcionamiento estático sólo obtiene el consumo para un momento determinado. Se comporta como el funcionamiento dinámico para una única iteración.

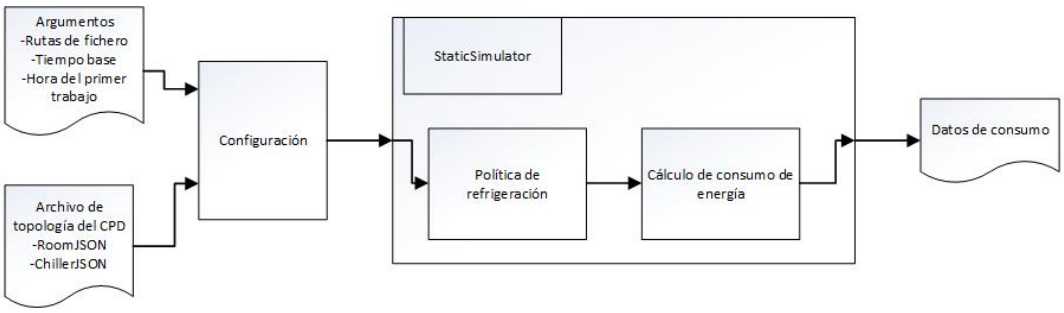

Figura 2.2.3.a. Funcionamiento estático de DC-Sim

- Funcionamiento dinámico: La simulación dinámica supone que el sistema evoluciona en función del tiempo, es decir, la temperatura a la que se ejecuta la simulación varía en función de un archivo de temperaturas y se van agregando y terminando tareas en los servidores a lo largo de dicha ejecución.

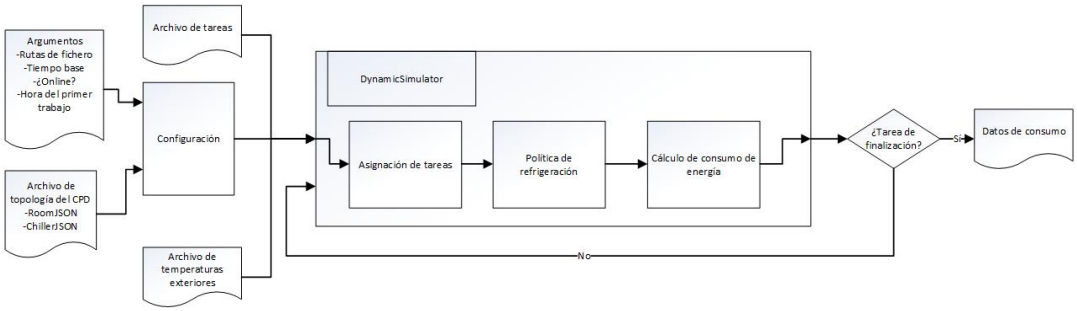

Figura 2.2.3.b. Funcionamiento dinámico de DC-Sim.

Las fases de ambos tipos de simulaciones se resumen en:

- Configuración: A partir de los datos del archivo de topología se construyen los modelos de los sistemas con los que trabaja el simulador, es decir la refrigeración y la sala de servidores; además se configuran algunos parámetros como los tiempos de inicio y de llegada de los trabajos.

- Asignación de tareas: Esta tarea sólo está disponible en modo dinámico y se encarga de asignar las tareas de un archivo a un servidor. La información de las tareas en el archivo es: su servidor objetivo, el tiempo durante el cual sé ejecutarán y cómo modifican el consumo dinámico de las CPUs y las memorias.

- Política de refrigeración: Esta etapa se encarga de modificar algunos parámetros de refrigeración cómo la velocidad de los ventiladores o la temperatura del aire que llega a los racks, según sea la política de enfriamiento empleada.

- Cálculo de consumo de energía: Esta fase ejecuta la lógica relacionada con el consumo y devuelve los valores de potencia y energía para un momento determinado.

2.2.4. Implementación de los modelos de consumo en DC-Sim

El simulador actual dispone de un modelo para cada uno de los elementos de consumo eléctrico de un data center refrigerado por un sistema de refrigeración In-Rack:

El modelo de consumo de data center considera la contribución de los k_1 IRCs, los k_2 servidores y el sistema de refrigeración (enfriador, bomba de agua y torre de refrigeración). La ecuación del modelo de datacenter sería:

$$P_{DataCenter} = \sum_{n=0}^{k_2} P_{server,n} + \sum_{n=0}^{k_1} P_{IRC,n} + P_{chiller} + P_{pump} + P_{CoolingTower} \tag{2.7}$$

- Servidor: El modelo de consumo de un servidor está definido por la suma de potencias:

$$P_{server} = P_{idle} + P_{cpu,dyn} + P_{leakeage} + P_{fan} + P_{mem,dyn} \tag{2.8}$$

  Siendo P_{idle} la potencia correspondiente a la cpu, el disco y las memorias en idle; $P_{cpu,dyn}$ y $P_{mem,dyn}$ los incrementos de consumo de potencia de la CPU y la memoria, respectivamente, al realizar una tarea; $P_{leakeage}$ será el consumo extra de potencia derivado del incremento de temperatura; y el P_{fan} vendrá de la potencia usada por el ventilador. Cabe recordar que estos valores variarán en función de las tareas y el servidor empleados.

- Refrigerador In-Rack (IRC): El consumo de potencia del IRC es función cúbica con relación al flujo de aire que recibe de los servidores, ya que debe extraer el calor de dicho flujo de aire mediante la ventilación del IRC y la circulación de agua fría. La ecuación del consumo de potencia del IRC es:

$$P_{IRC} = \alpha_3 * airflow^3 + \alpha_2 * airflow^2 + \alpha_1 * airflow + \alpha_0 \tag{2.9}$$

  Siendo $airflow$ la suma del flujo de aire de los servidores refrigerados por el IRC correspondiente, y α_3, α_2, α_1, α_0 coeficientes de la ecuación y específicos del IRC en cuestión.

- Enfriador, torre de refrigeración y bomba de agua: El modelo de consumo del sistema de refrigeración tendrá en cuenta la temperatura exterior. Si ésta es suficientemente baja se aplica el modelo de consumo freecooling, en el que $P_{chiller} = 0$; el $P_{CoolingTower}$ se obtiene de una lookup table (LUT) creada a partir de las datasheets facilitadas por el fabricante. Si no se pudiese aplicar una política de freecooling, $P_{chiller}$ vendrá dado por las siguiente ecuación:

$$P_{chiller} = \frac{Q_{evap}}{COP_{chiller}} \tag{2.10}$$

siendo Q_{evap} el sumatorio del calor generado por los servidores enfriados por cada IRC. $COP_{chiller}$ es el coeficiente de rendimiento del enfriador:

$$COP_{chiller} = \frac{T_{evap}}{Tcond - T_{evap}} \tag{2.11}$$

donde T_{evap} es la temperatura del agua en el evaporador y T_{cond} la temperatura en el condensador. Finalmente, P_{pump} es proporcional al flujo de agua del IRC y al cambio de presión e inversamente proporcional a la eficiencia de la bomba:

$$P_{pump} = \frac{f_{IRCwater} * \Delta P}{\epsilon_{pump}} \tag{2.12}$$

2.2.5. DC-Simulator en xDEVS

El objetivo de trasladar DC-Sim a SFIDE implica la formalización de un modelo en DEVS a partir del existente, el cual es ad-hoc. Las ventajas de esta reformalización son:

1. Facilidad de comunicación con otros sistemas DEVS.

2. Sustancial mejora de la gestión del tiempo para los eventos de simulación.

3. Posibilidad de paralelización de los eventos, con la consecuente reducción del tiempo de ejecución en topologías con excesiva carga de trabajo.

En el nuevo diseño para DEVS los elementos de consumo de DC-Sim han sido sustituidos por componentes de DEVS. Cada nuevo componente dispone de puertos para comunicar la información que se intercambiaban en DC-Sim. La figura 2.2.5.1 representa un boceto de la estructura en DEVS.

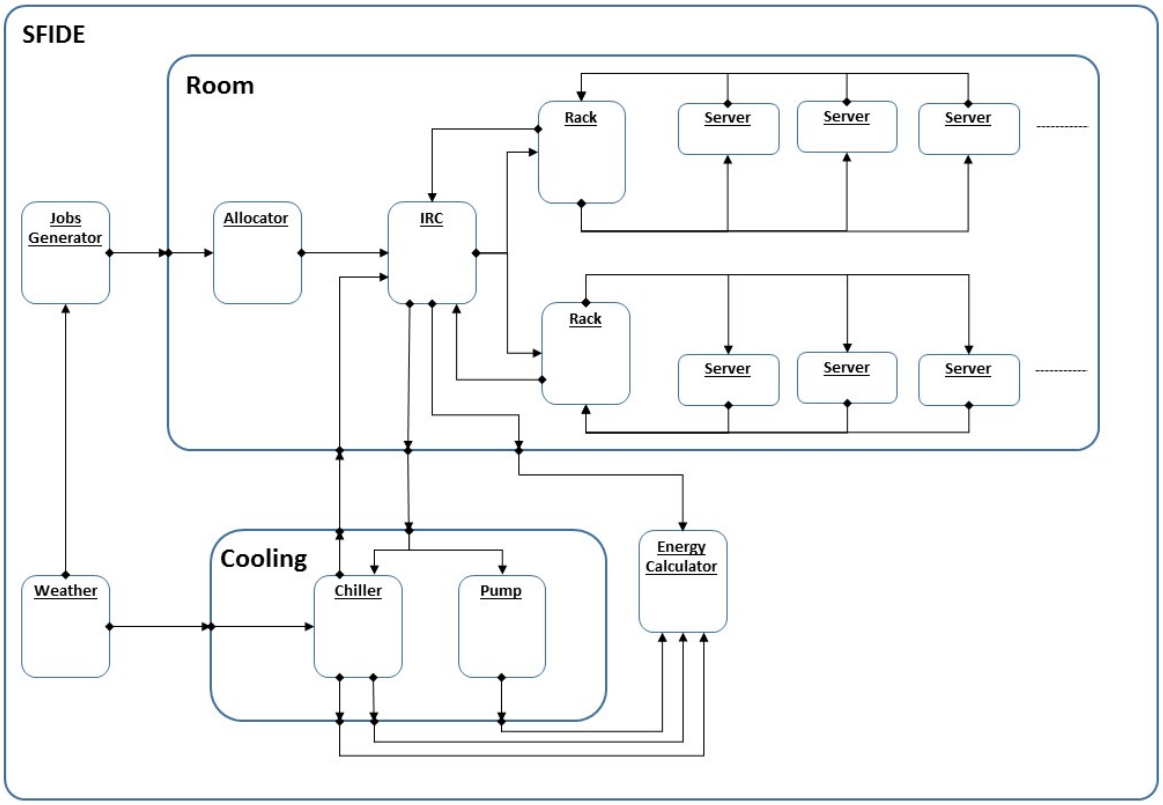

Figura 2.2.5.1: Estructura DEVS

Capítulo 3

Estructura y comportamiento

La estructura del modelo DEVS consta de la carga de trabajo, gestionada por el elemento atómico ***Jobs Generator***, la medida de la temperatura exterior, gestionada por el elemento atómico ***Weather***, una serie de elementos acoplados ***Rooms***, así como el sistema de enfriamiento ***Cooling***, con el cual estan asociadas todas las ***Rooms***. Los datos generados por los diferentes componentes son recibidos por el atómico ***Energy Calculator***, que se encarga de generar la salida correspondiente a la simulación.

3.1. Estructura

3.1.1. SFIDE: modelo acoplado raíz

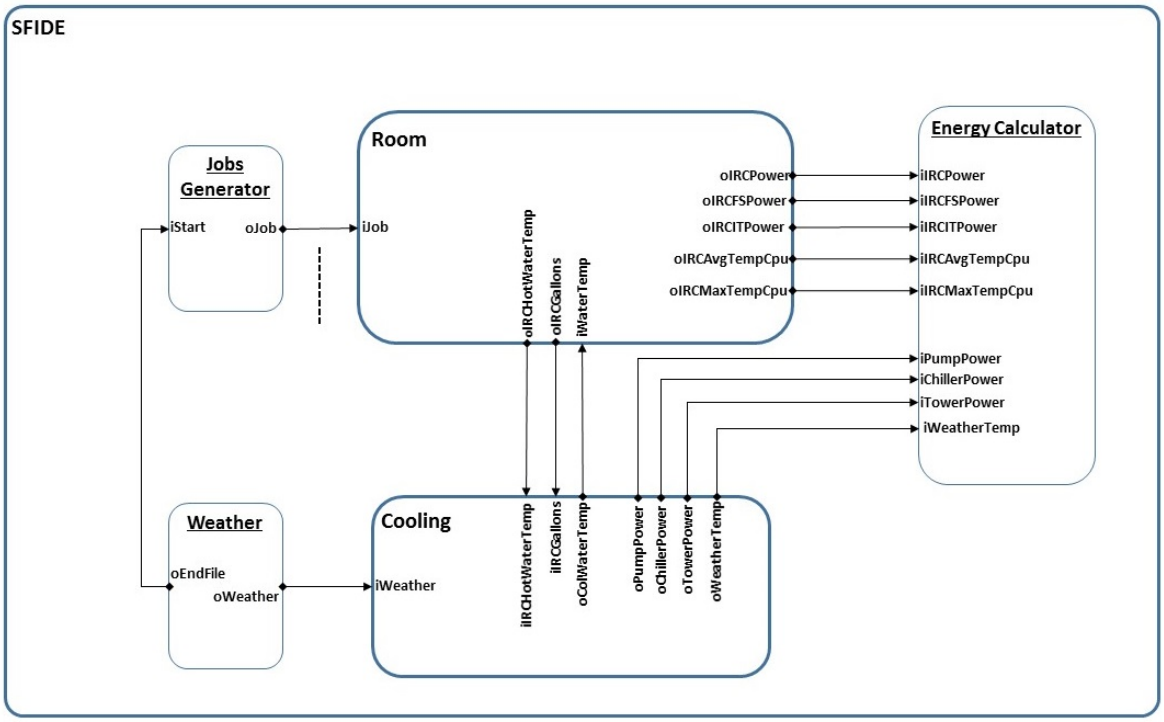

Figura 3.1.1.1: SFIDE, el modelo acoplado raíz.

El modelo acoplado ***SFIDE***, de acuerdo con la definición formal expresada en 2.6, está formado por los siguientes componentes:

- **Jobs Generator**: Este componente atómico va distribuyendo los diferentes jobs generados a partir del fichero de entrada. A partir de éste, se genera la estructura del ***job***, con los siguientes atributos:
  - ◇ **id:** Identificador del job.
  - ◇ **begin:** Indica si el job ha finalizado o no.
  - ◇ **startTime:** Hora *("tiempo")* de inicio.
  - ◇ **endTime:** Hora *("tiempo")* de finalización.
  - ◇ **ircName:** Nombre del IRC en el que se encuentra el servidor destinatario del job.
  - ◇ **rackName:** Nombre del Rack en el que se encuentra el servidor destinatario del job.
  - ◇ **serverName:** Nombre del Server destinatario del job.
  - ◇ **numThreads:** Número de threads empleados para el job.
  - ◇ **numCores:** Número de cores de la cpu.
  - ◇ **cpuPower:** Potencia dinámica de la cpu.
  - ◇ **memPower:** Potencia dinámica de la memoria.

  Tiene configurado un puerto de entrada ***"iStart"***, a través del cual el componente atómico ***Weather*** le indica si tiene que enviar nuevos jobs. Tiene también configurado un puerto de salida, ***"oOut"***, a través del cual se transmiten al componente ***Room*** los jobs generados.

  En la inicialización lee el primer *job* del fichero ***joblogger***, modifica su estado y se ejecuta la función λ, que coloca el *job* en el puerto *"oOut"* para ser transmitido al componente ***Allocator***.

- **Room**: Existen *R* componentes acoplados ***Room***. Se definirá en la sección 3.1.2.

- **Weather**: Componente atómico que lee desde el fichero que contiene las temperaturas exteriores de simulación para ser enviadas al componente acoplado ***Cooling***, a través del puerto de salida ***"oWeather"***. Cuando finaliza el fichero, a través del puerto de salida ***"oEndfile"*** se envía el mensaje oportuno a ***Jobs Generator*** para que éste cambie su estado a *"passive"* y finalice la simulación.

  En la inicialización lee el primer valor de temperatura del fichero *Weather.txt*, modifica su estado y se ejecuta la función λ, que coloca el valor leído en el puerto *"oOut"* para ser transmitido al componente ***Chiller***.

- **Cooling**: Elemento acoplado que modela el sistema de refrigeración líquida de los distintos ***IRC***. Se definirá en la sección 3.3.

- **Energy Calculator**: Componente atómico que ejerce como transductor de los datos de consumo del Data Center durante la simulación, así como de las temperaturas máxima y media alcanzadas por los servidores que lo constituyen. También recibe la temperatura externa.

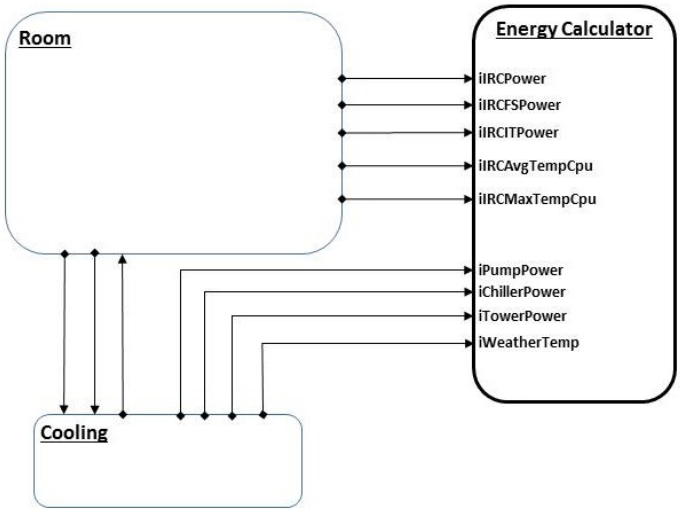

Figura 3.1.1.2: Modelo atómico Energy Calculator.

El modelo acoplado ***SFIDE*** no contiene ninguna entrada ni salida ***(X=Y=∅)***, ni conexiones externas de entrada o salida ***(EIC=EOC=∅)***. Contiene las siguientes conexiones internas ***IC***:

Jobs Generator	oJob → iJob	**Room**
Weather	oWeather → iWeather	**Cooling**
Room	oIRCHotWater → iIRCHotWaterTemp	**Cooling**
	oIRCGallons → iIRCGallons	
Room	oIRCPower → iIRCPower	**Energy Calculator**
	oIRCFSPower → iIRCFSPower	
	oIRCITPower → iIRCITPower	
	oIRCAvgTempCpu → iIRCAvgTempCpu	
	oIRCMaxTempCpu → iIRCMaxTempCpu	
Cooling	oColdWaterTemp → iWaterTemp	**Room**
Cooling	oPumpPower → iPumpPower	**Energy Calculator**
	oChillerPower → iChillerPower	
	oTowerPower → iTowerPower	
	oWeatherTemp → iWeatherTemp	

Cuadro 3.1: Conexiones internas de SFIDE

3.1.2. Room

La figura 3.1.2.1 muestra el modelo acoplado Room. Un Data Center contiene *R* componentes de este tipo.

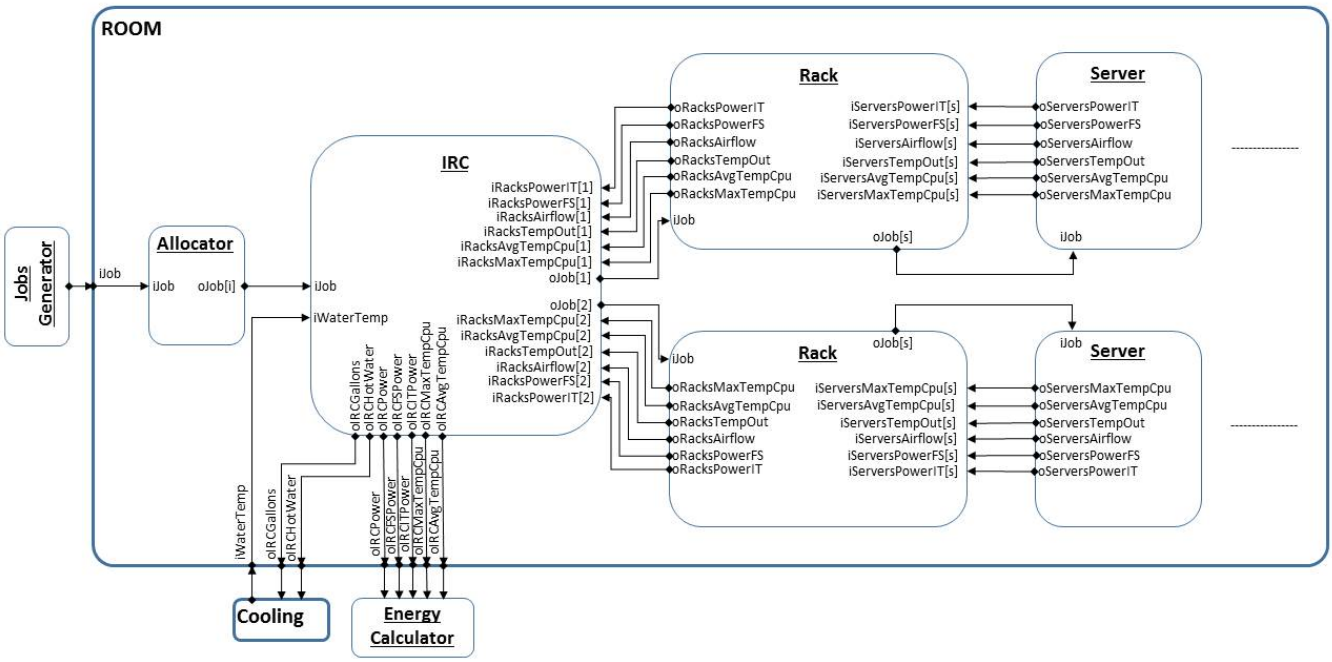

Figura 3.1.2.1: Modelo acoplado Room.

El modelo acoplado Room consta de los siguientes componentes:

- **Allocator**: Cada componente acoplado Room consta de un componente atómico Allocator. A través del puerto de entrada ***"iJob"*** recibe los trabajos procedentes del atómico ***Jobs Generator***. Dispone de tantos puertos de salida ***"oJob"*** como IRCs constituyan el componente ***Room***. Su función es asignar el trabajo recibido al IRC, Rack y Server de manera que se optimice el consumo energético.

- **IRC**: In-Rack-Cooling, componente atómico que modela el comportamiento del elemento con la función de refrigerar el calor generado por los servidores alojados en los dos racks asociados a éste.

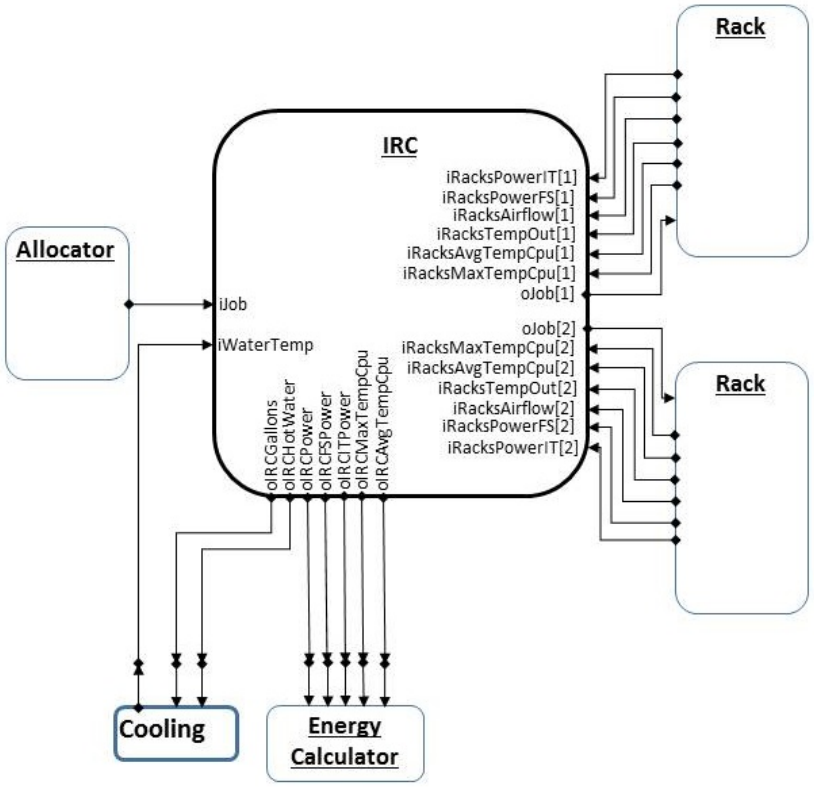

Figura 3.1.2.2: Modelo atómico IRC.

Cada componente ***Room*** contiene *I* componentes atómicos ***IRC***.

A través del puerto de entrada ***"iJob"*** recibe los trabajos procedentes del atómico ***Jobs Generator*** y los transmite a través del puerto de salida ***"oJob"*** correspondiente al rack al que está asignado.

Dispone de cinco puertos de entrada por cada uno de los racks, a través de los cuales recibe:

- ◇ A través del puerto ***"iRacksPowerIT"*** recibe la potencia consumida por las CPU's, memoria y disco de los servidores.
- ◇ A través del puerto ***"iRacksPowerFS"*** recibe la potencia consumida por los ventiladores de los servidores.
- ◇ A través del puerto ***"iRacksAirflow"*** recibe el flujo de aire generado por los ventiladores de los servidores de cada rack.
- ◇ A través del puerto ***"iRacksTempOut"*** recibe la temperatura del aire, incrementada por el calor generado por los servidores.
- ◇ A través del puerto ***"iRacksAvgTempCpu"*** recibe la temperatura media de las CPU's.
- ◇ A través del puerto ***"iRacksMaxTempCpu"*** recibe la temperatura máxima de las alcanzadas en una de las CPU's.

Por el puerto de entrada ***"iWaterTemp"*** recibe la temperatura del agua procedente del componente ***Cooling*** que servirá para la refrigeración.

Procesando los datos recibidos, genera las siguientes salidas:

- ◇ A través del puerto ***"oIRCGallons"*** transmite el flujo de agua necesario para la refrigeración.
- ◇ A través del puerto ***"oIRCHotWater"*** transmite la temperatura del agua de retorno a **Cooling**.
- ◇ A través del puerto ***"oIRCPower"*** transmite la potencia consumida por el **IRC**.
- ◇ A través del puerto ***"oIRCAvgTempCpu"*** transmite la temperatura media de las CPU's.
- ◇ A través del puerto ***"oIRCMaxTempCpu"*** transmite la temperatura máxima de las alcanzadas en las CPU's.

- **Rack**: El componente atómico ***Rack*** tiene dos funciones. Por una parte, recibe los trabajos a ejecutar en cada uno de los ***Server*** alojados en él, transmitiéndoselos a través de los puertos de salida correspondientes. Por otra, recibe los datos generados por cada uno de los ***Server*** para generar las salidas que transmitirá al ***IRC***.

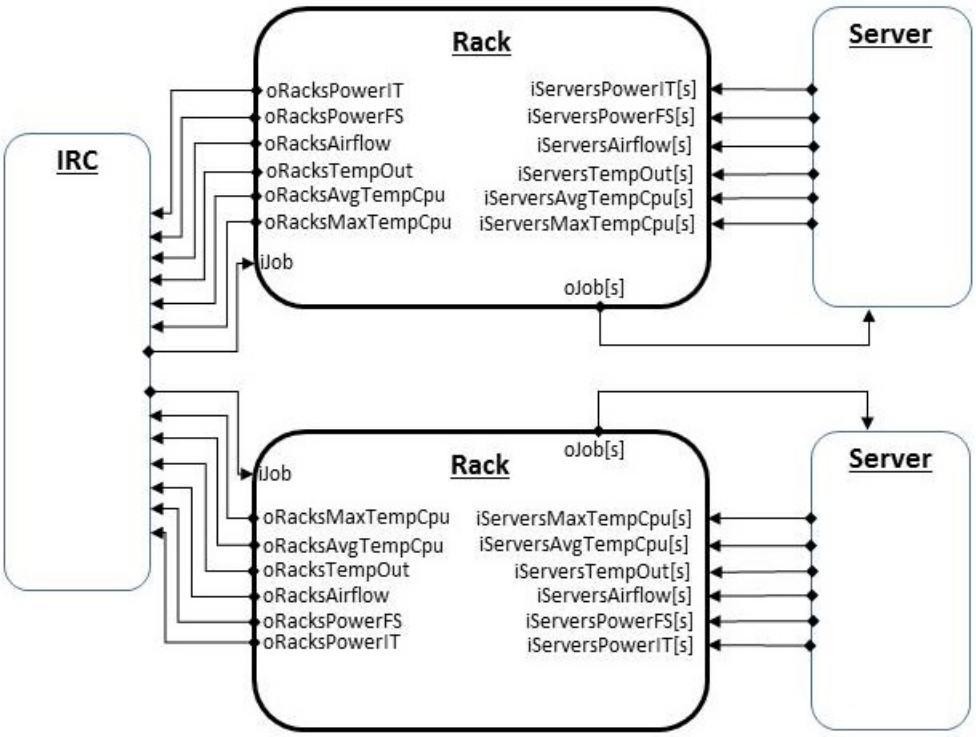

Figura 3.1.2.3: Modelo atómico Rack.

A través del puerto de entrada ***"iJob"*** recibe los trabajos procedentes del componente ***IRC*** y los transmite a través del puerto de salida ***"oJob"*** correspondiente al server al que está

asignado. Este puerto de salida es realmente un "map"de puertos, existiendo uno por cada ***Server*** que se encuentra alojado en cada ***Rack***.

Están configurados seis "maps"de puertos de entrada para recibir de cada uno de los ***Server*** los datos generados por éstos:

- ◇ A través del puerto ***"iServersPowerIT"*** recibe la potencia consumida por las CPU's, memoria y disco del ***Server*** correspondiente.
- ◇ A través del puerto ***"iServersPowerFS"*** recibe la potencia consumida por sus ventiladores.
- ◇ A través del puerto ***"iServersAirflow"*** recibe el flujo de aire generado por sus ventiladores.
- ◇ A través del puerto ***"iServersTempOut"*** recibe la temperatura del aire de cada servidor.
- ◇ A través del puerto ***"iServersAvgTempCpu"*** recibe la temperatura media de las cpus contenidas en el servidor.
- ◇ A través del puerto ***"iServersMaxTempCpu"*** recibe la temperatura máxima alcanzada por alguna de las cpus contenidas en el servidor.

Para transmitir los datos generados a partir de la información recibida de los servidores dispone de seis puertos de salida:

- ◇ A través del puerto ***"oRackPowerIT"*** transmite la potencia dinámica, de memoria y de disco consumida por los ***Server***.
- ◇ A través del puerto ***"oRackPowerFS"*** transmite la suma de potencias consumidas por los ventiladores del conjunto de servidores.
- ◇ A través del puerto ***"oRackAirflow"*** transmite el flujo de aire generado por los ventiladores de todos los servidores del Rack.
- ◇ A través del puerto ***"oRackTempOut"*** transmite la temperatura del aire.
- ◇ A través del puerto ***"oRackAvgTempCpu"*** transmite la temperatura media de todas las cpus.
- ◇ A través del puerto ***"oRackMaxTempCpu"*** transmite la temperatura máxima alcanzada en las cpus.

- **Server**: El componente atómico ***Server*** es el encargado de calcular la potencia consumida por los elementos de proceso, la memoria, el disco y los ventiladores, así como el flujo de aire y su temperatura de salida y las temperaturas máximas y medias de las cpus. Cada componente ***Rack*** se comunica con *S* ***Server***.

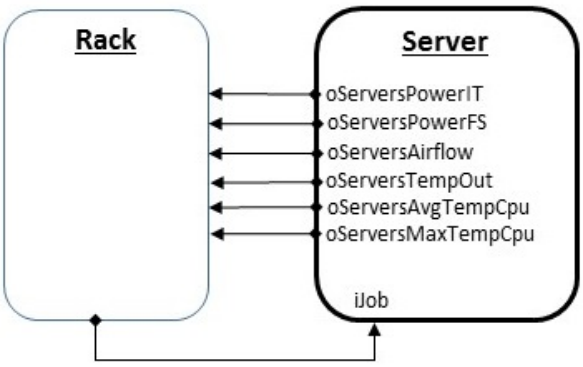

Figura 3.1.2.4: Modelo atómico Server.

A través del puerto de entrada ***"iJob"*** recibe los trabajos procedentes componente ***Rack***, calculando en función de éstos los parámetros que transmitirá de nuevo a ***Rack*** a través de los puertos de salida:

- ◇ El puerto ***"oServerPowerIT"*** transmite la potencia dinámica, de memoria y de disco consumida por el servidor.
- ◇ El puerto ***"oServerPowerFS"*** transmite la potencia consumida por los ventiladores.
- ◇ El puerto ***"oServerAirflow"*** transmite el flujo de aire generado por los ventiladores.
- ◇ El puerto ***"oServerTempOut"*** transmite la temperatura del aire expulsado por el servidor.
- ◇ El puerto ***"oServerAvgTempCpu"*** transmite la temperatura media alcanzada por las cpus que contiene el servidor.
- ◇ El puerto ***"oServerMaxTempCpu"*** transmite la máxima temperatura alcanzada por alguna de las cpus.

En la inicialización de ***Server*** se modifica su estado y al ejecutarse λ los valores de inicialización se colocan en los puertos correspondientes para su transmisión al componente ***Rack***.

Como refleja la figura 3.1.2.1, el modelo acoplado ***Room*** contiene dos conexiones de entrada externa ***(EIC)***:

- A través del puerto ***"iJob"*** se reciben los jobs procedentes de ***"Jobs Generator"*** con destino a ***"Allocator"***.
- A través del puerto ***"iWaterTemp"*** se recibe la temperatura exterior, procedente de ***"Cooling"*** con destino a ***"IRC"***.

Las conexiones de salida externa ***(EOC)*** son siete.

- Dos puertos conectan con el componente ***Cooling***:

- ◇ El puerto ***"oIRCGallons"*** se emplea para la transmisión de la cantidad de agua necesaria para la refrigeración.

- ◇ A través del puerto ***"oIRCHotWater"*** se transmite la temperatura del agua que circula a través del IRC.

- Cinco puertos conectan con el componente ***Energy Calculator***:

  - ◇ A través del puerto ***"oIRCPower"*** se transmite la potencia consumida por el ***IRC***.

  - ◇ A través del puerto ***"oIRCFSPower"*** se transmite la suma de las potencias consumidas por los ventiladores de todos los ***Server*** refrigerados a través del ***IRC***.

  - ◇ A través del puerto ***"oIRCITPower"*** se transmite la suma de las potencias consumidas por los ***Server***.

  - ◇ A través del puerto ***"oIRCMaxTempCpu"*** se transmite la máxima temperatura alcanzada por alguno de los ***Server***.

  - ◇ A través del puerto ***"oIRCAvgTempCpu"*** se transmite la temperatura media alcanzada en los ***Server*** del correspondiente ***IRC***.

Para conectar los diferentes componentes,consta de las siguientes conexiones internas ***IC***:

Allocator	oJob → iJob	**IRC**
IRC	oJob → iJob	**Rack**
Rack	oJob → iJob	**Server**
Server	oServerPowerIT → iServersPowerIT	**Rack**
	oServerPowerFS → iServersPowerFS	
	oServerAirflow → iServersAirflow	
	oServerTempOut → iServersTempOut	
	oServerAvgTempCpu → iServersAvgTempCpu	
	oServerMaxTempCpu → iServersMaxTempCpu	
Rack	oRackPowerIT → iRacksPowerIT	**IRC**
	oRackPowerFS → iRacksPowerFS	
	oRackAirflow → iRacksAirflow	
	oRackTempOut → iRacksTempOut	
	oRackAvgTempCpu → iRacksAvgTempCpu	
	oRackMaxTempCpu → iRacksMaxTempCpu	

Cuadro 3.2: Conexiones internas de Room

3.1.3. Cooling

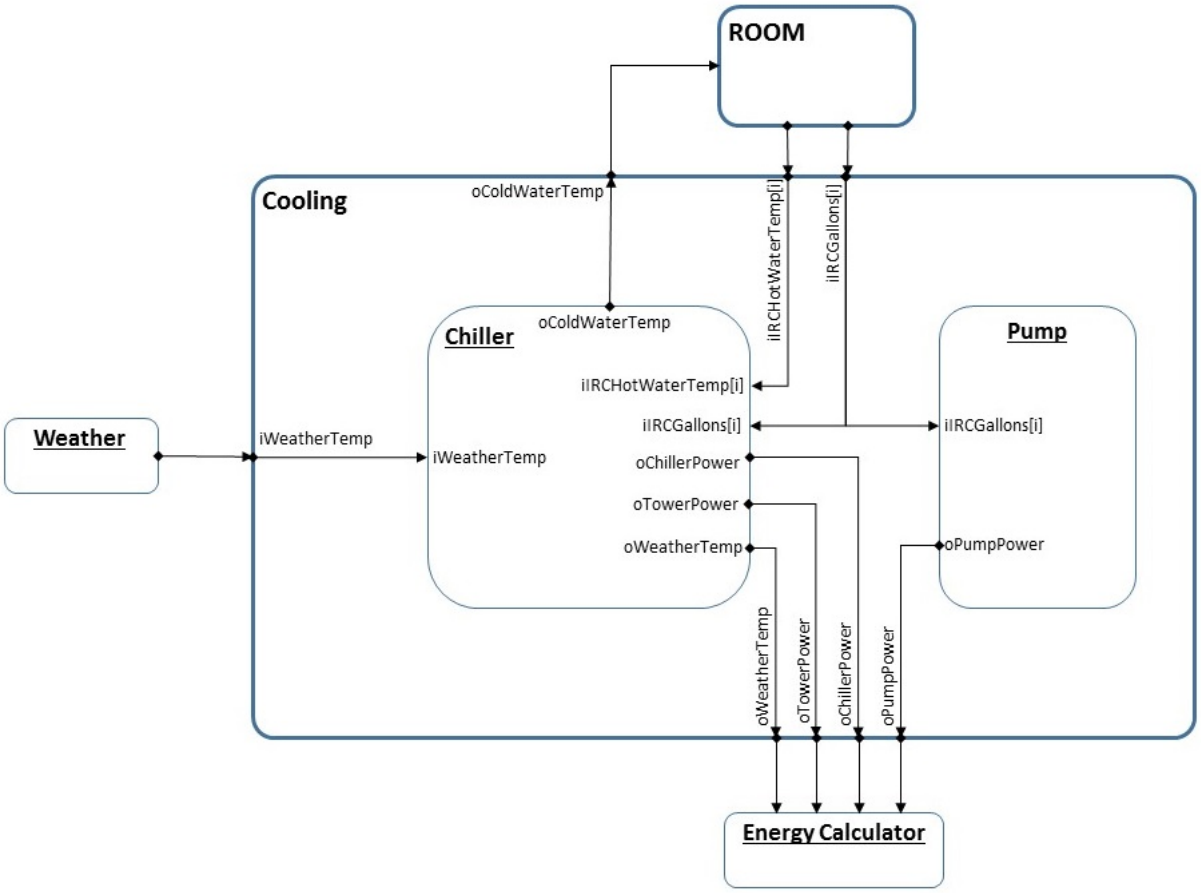

Figura 3.1.3.1: Modelo acoplado Cooling.

La figura 3.1.3.1 muestra el modelo acoplado ***Cooling***, que está formado por los siguientes componentes atómicos:

- **Chiller**: Tiene la función de calcular el consumo de potencia del sistema de enfriamiento del Data Center a partir de los datos recibidos del componente ***Room*** y la temperatura exterior suministrada por el componente ***Weather***. Dispone de los siguientes puertos:
  - ◇ A través del puerto ***"iWeatherTemp"*** recibe la temperatura exterior.
  - ◇ A través de los puertos ***"iIRCHotWaterTemp"*** e ***"iIRCGallons"*** recibe la temperatura y flujo de agua de cada IRC que es necesario refrigerar.
  - ◇ A través del puerto ***"oColdWaterTemp"*** transmite la temperatura del agua que sirve el ***Chiller***.
  - ◇ A través del puerto ***"oWeatherTemp"*** transmite a ***Energy Calculator*** la temperatura exterior para generar el informe de la simulación.

- ◇ A través de los puertos ***"oChillerPower"*** y ***"oTowerPower"*** transmite a ***Energy Calculator*** la potencia consumida por el enfriador y la torre de refrigeración, respectivamente.

Cuando se inicializa se cambia de estado y se ejecuta λ, que coloca en el puerto *"oColdWaterTemp"* la temperatura del agua fría para ser utilizada por el componente ***IRC***.

- **Pump**: A partir del flujo de agua que es necesario hacer circular para la refrigeración, recibido a través del puerto ***"iIRCGallons"***, calcula la potencia consumida por la bomba, transmitiendo el dato a ***Energy Calculator*** a través del puerto ***"oPumpPower"***

Como refleja la figura 3.1.3.1, el modelo acoplado ***Cooling*** presenta tres conexiones de entrada externa ***(EIC)***:

- A través del puerto ***"iWeatherTemp"*** recibe la temperatura exterior desde el componente ***Weather*** para ser utilizado por ***Chiller***.
- A través del puerto ***"iIRCHotWaterTemp"***, conectado con ***Chiller***, recibe la temperatura del agua de cada IRC.
- A través del puerto ***"iIRCGallons"*** recibe el fujo de agua de cada IRC, que es transmitido tanto a ***Chiller*** como a ***Pump***.

Tiene una conexión de salida externa ***(EOC)*** que conecta con el componente ***Room***, ***"oColdWaterTemp"***, para transmitir la temperatura del agua servida por ***Chiller***, y cuatro que conectan con ***Energy Calculator***:

- A través del puerto ***"oWeatherTemp"*** transmite la temperatura exterior.
- A través del puerto ***"oTowerPower"*** transmite la potencia consumida por la torre de refrigeración.
- A través del puerto ***"oChillerPower"*** transmite la potencia consumida por el enfriador.
- A través del puerto ***"oPumpPower"*** transmite la potencia consumida por la bomba.

Entre los componentes atómicos del acoplado ***Cooling*** no existe ninguna conexión, por lo tanto éste no tiene ninguna conexión interna ***(IC=$\emptyset$)***.

3.2. Comportamiento

El comportamiento de los componentes atómicos está controlado por las funciones:

- δ_{int}, función de transición interna.
- δ_{ext}, función de transición externa.
- λ, función de salida.

· δ_{con}, función de transición confluyente. En todos los componentes atómicos, esta función establece que se ejecute en primer lugar la función δ_{int} y posteriormente δ_{ext} con *e=0*, cuando se produzca la confluencia de una transición interna con una externa.

· ta, función de avance de tiempo, que devuelve siempre el valor de σ.

3.2.1. Jobs Generator

Función de transición interna: Lee del fichero de *joblogger* los datos del siguiente job a ejecutar y cambia el estado durante el tiempo que debe que transcurrir antes de transmitir el job leído.

Función de transición externa:No está implementado ningún comportamiento porque no se ha contemplado que reciba nada.

Función de salida: Coloca el job en el puerto *"oOut"* para ser leído por el componente *Allocator*.

3.2.2. Allocator

Función de transición interna: Inicializa a *"0"* el job actual y cambia el estado a *"passive"*.

Función de transición externa: Lee del puerto *"iJob"* el job a ejecutar y cambia el estado a *"active"* con *ta=0.0*.

Función de salida: Coloca el job en el puerto de salida correspondiente al *IRC* al que está asignado el job.

3.2.3. IRC

Función de transición interna: Inicializa a *0.0* todas las variables y cambia el estado a *"passive"*.

Función de transición externa: Lee del puerto *"iJob"* el job a ejecutar y cambia el estado a *"active"* con *ta=0.0*. Lee todos los puertos de entrada a través de los cuales recibe los datos procedentes de los dos *Racks*, así como el puerto por el que recibe la temperatura del agua, procedente de *Chiller*. Cuando dispone de todos los datos cambia de estado a *"active"* con *ta=0.0*.

Función de salida: Coloca el job en el puerto de salida correspondiente al *Rack* al que está asignado el job. Coloca los datos generados a partir de las entradas en los puertos de salida correspondientes para ser transmitidos a *Cooling* y *Energy Calculator*.

El consumo de potencia del *IRC* es función del flujo de aire que recibe de los servidores, ya que debe extraer el calor de dicho flujo de aire mediante la ventilación del *IRC* y la circulación del agua fría. El cálculo se realiza mediante la ecuación:

$$P_{IRC} = \alpha_3 * airflow^3 + \alpha_2 * airflow^2 + \alpha_1 * airflow + \alpha_0 \tag{3.1}$$

donde:

· *airflow* es la suma del flujo de aire de los servidores refrigerados por el *IRC*.

· α_0, α_1 , α_2 y α_3 los coeficientes de regresión específicos del *IRC* correspondiente. En este caso: 198.9612, -0.0317, $3{,}0196 * 10^{-5}$ y $1{,}5658 * 10^{-8}$, respectivamente.

3.2.4. Rack

Función de transición interna: Inicializa a *0.0* todas las variables y cambia el estado a *"passive"*.

Función de transición externa: Lee del puerto *"iJob"* el job a ejecutar y cambia el estado a *"active"* con *ta=0.0*. Lee todos los puertos de entrada a través de los cuales recibe los datos procedentes de los *Server*. Cuando dispone de todos los datos cambia el estado a *"active"* con *ta=0.0*.

Función de salida: Coloca el job en el puerto de salida correspondiente al *Server* al que está asignado el job. Coloca los datos generados a partir de las entradas en los puertos de salida correspondientes para ser transmitidos a *IRC*.

3.2.5. Server

Función de transición interna: Inicializa a *0* a variable currentJob y cambia el estado a *"passive"*.

Función de transición externa: Lee del puerto *"iJob"* el job a ejecutar y después de realizar los cálculos de consumo energético y temperatura de los distintos componentes (función *updateState()*), cambia el estado a *"active"* con *ta=0.0*.

Función de salida: Coloca los datos generados en los puertos de salida correspondientes para ser transmitidos al *Rack*.

El componente *Server* genera seis salidas:

- ***serverPowerIT***, que se obtiene mediante la ecuación:

$$serverPowerIT = idlePower + memPower + cpuPower + leakagePower \quad (3.2)$$

  donde *memPower* y *cpuPower* son la potencia dinámica de la memoria y cpu, que se obtiene de los atributos del job que se está ejecutando, *idlePower* es la potencia consumida en reposo y *leakagePower* es el consumo debido al incremento de la temperatura durante la ejecución de la carga de trabajo:

$$leakagePower = \alpha_0 + \alpha_1 * T_{CPU} + \alpha_2 * T_{CPU}^2 \quad (3.3)$$

  donde T_{CPU} es la temperatura de la CPU y α_0, α_1 y α_2 coeficientes de regresión.

- ***serverPowerFS***, la potencia consumida por el ventilador, que se calcula en función de la velocidad.

- ***serverAirflow***, el flujo de aire, también en función de la velocidad del ventilador.

- ***serverTempOut***, la temperatura del aire a la salida del *Server*, que es directamente proporcional a la temperatura de *inlet* y a *serverPowerIT*, e inversamente proporcional al flujo de aire generado (*serverAirflow*).

- ***serverAvgTempCpu***, la temperatura media de las CPU's. La temperatura se calcula en función de la temperatura de *inlet*, la velocidad del ventilador y la potencia dinámica de la CPU.

- ***serverMaxTempCpu***, la máxima temperatura alcanzada por una de las CPU's del *Server*.

3.2.6. Weather

Función de transición interna: Lee del fichero de temperaturas el siguiente registro, si existe, y cambia el estado a *"active"*, si quedan más registros por leer, o a *"pasive"*, si ya ha finalizado.

Función de transición externa: No está implementado ningún comportamiento porque no se ha contemplado que reciba nada.

Función de salida:Si se ha llegado al final del fichero de temperaturas, coloca en el puerto *"oStop"* el booleano *false* para ser leído por *Job Generator*. En caso contrario, coloca el valor de la temperatura leído en el puerto *"oOut"* para ser leído por el componente *Chiller*.

3.2.7. Chiller

Función de transición interna: Inicializa todas las variables y cambia el estado a *"passive"*.

Función de transición externa: Lee todos los puertos de entrada a través de los cuales recibe los datos procedentes de los *IRC*, así como el puerto del que recibe la temperatura exterior desde el componente *Weather*. Cuando dispone de todos los datos cambia el estado a *"active"* con *ta=0.0*.

Función de salida: Coloca en el puerto *"oColdWaterTemp"* la temperatura del agua para ser leída por *IRC*. En los puertos *"oChillerPower"*, *"oTowerPower"* y *"oWeatherTemp"*, respectivamente, los valores de consumo de Chiller y Tower, así como la temperatura exterior, para ser leídos por *Energy Calculator*.

3.2.8. Pump

Función de transición interna: Inicializa las variables y cambia el estado a *"passive"*.

Función de transición externa: Lee el dato de flujo de agua necesario para la refrigeración de cada uno de los *IRC* y en función de aquél calcula la potencia consumida en el proceso de bombeo, tras lo cual cambia el estado a *"active"* con *ta=0.0*.

Función de salida: Coloca en el puerto *"oPumpPower"* el dato de potencia calculado anteriormente.

3.2.9. Energy Calculator

Función de transición interna: Inicializa las variables y cambia el estado a *"passive"*.

Función de transición externa: Lee los datos de potencia consumida facilitados por los componentes *IRC*, *Chiller* y *Pump*, así como la temperatura externa,y los escribe en el fichero de salida.

Función de salida: Al no existir comunicación hacia ningún otro componente no es necesario que esté definida.

Capítulo 4

Resultados, conclusiones y futuros desarrollos

A continuación se explicarán los resultados del nuevo modelo DEVS, comparándolos con el modelo del simulador antiguo, al ser ambos resultados iguales para los diversos escenarios, queda validada la nueva implementación del modelo. Una vez contrastados, se explicarán las características de los resultados y las conclusiones extraídas de estos. Finalmente se explicarán las distintas posibilidades de ampliación.

4.1. Escenarios

Para las comparativas se emplearán dos configuraciones del CPD distintas y tres escenarios de temperatura (altas, moderadas y bajas):

- ◇ Temperaturas altas (12^{o}C a 29^{o}C)
- ◇ Temperaturas moderadas (-1^{o}C a 19^{o}C)
- ◇ Temperaturas bajas (-20^{o}C a 2^{o}C)

- Configuración mediana:
  - Configuración del data center: 9 IRC's, 18 racks y 324 servidores DELL.
  - Conjunto de trabajos: 1.503.350 trabajos.
- Configuración pequeña:
  - Configuración del data center: 3 IRC's, 6 racks y 108 servidores DELL.
  - Conjunto de trabajos: 843.109 trabajos.

4.2. Resultados

En la gráficas que se muestran a continuación se representan los siguientes parámetros:

- **IT**. Se corresponde con el consumo de las unidades de cómputo dentro de los servidores. Varía debido al número y tipo de servidores, así como a la carga de trabajo.

- **FS**. Simboliza el consumo de la refrigeración por aire de dentro de cada servidor y depende de la velocidad de los ventiladores. Al emplearse una política de enfriamiento constante este consumo es constante.

- **IRC**. Es el consumo referido a los ventiladores que impulsan el aire frío de los conductos de refrigeración a los racks. La velocidad de los ventiladores es constante y por tanto su consumo también lo es.

- **cooling**. Se refiere a la suma de consumos de la bomba, la torre y el sistema de refrigeración (chiller).

- **outdoorT**. Es la temperatura exterior.

- **AvgTemp**. Representa la media de temperaturas alcanzadas por los servidores.

- **MaxTemp**. Representa la máxima temperatura alcanzada por alguno de los servidores durante el periodo de simulación.

4.2.1. Resultados escenario pequeño

- ***Temperatura fría***

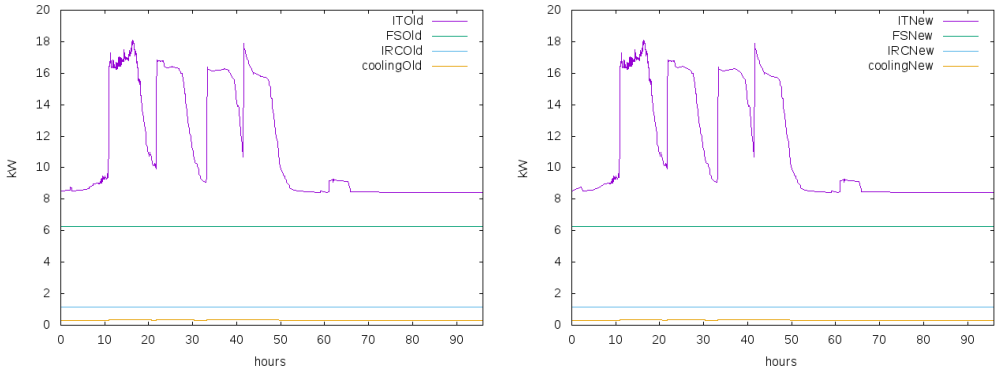

Figura 4.2.1.a: Consumo por componentes DC-Sim (izquierda) y SFIDE(derecha) en el escenario pequeño con temperaturas frías. En ambos modelos se obtienen valores iguales para todos los componentes: IT, FS, IRC y Cooling.

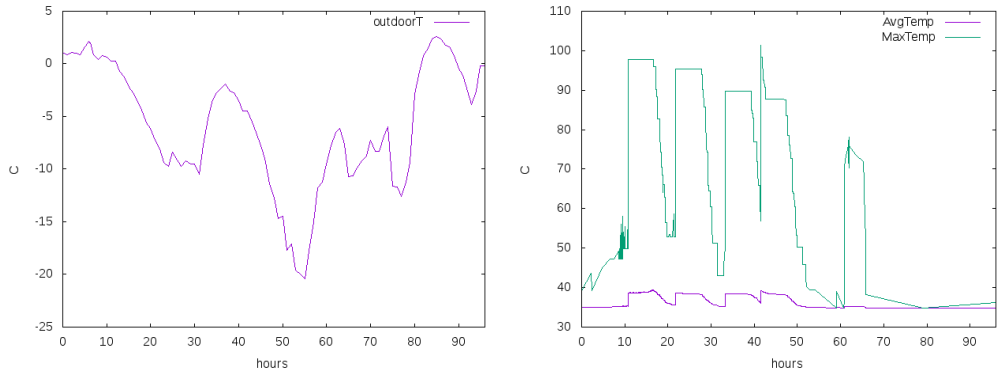

Figura 4.2.1.b: Temperatura exterior (izquierda) y de los servidores (derecha). La temperatura en los servidores está condicionada por la diferente carga de trabajo en cada instante.

Como se observa en la *figura 4.2.1.a*, el consumo producido por el sistema de refrigeración se mantiene constante durante todo el período, y con un valor de aproximadamente 0,32kW. Esta circunstancia es debido a las bajas temperaturas en el exterior *(figura 4.2.1.b(izquierda))*, que permiten la refrigeración por *freecooling*. La práctica totalidad del consumo de *cooling* es debido al funcionamiento de la *bomba*, siendo inapreciables los consumos del *chiller* y *tower*. Se observa la similitud de datos obtenidos en ambos simuladores.

La evolución de las temperaturas media y máxima de los servidores, representadas en la *figura 4.2.1.b*, está relacionada con el consumo de *IT*.

- ***Temperatura moderada***

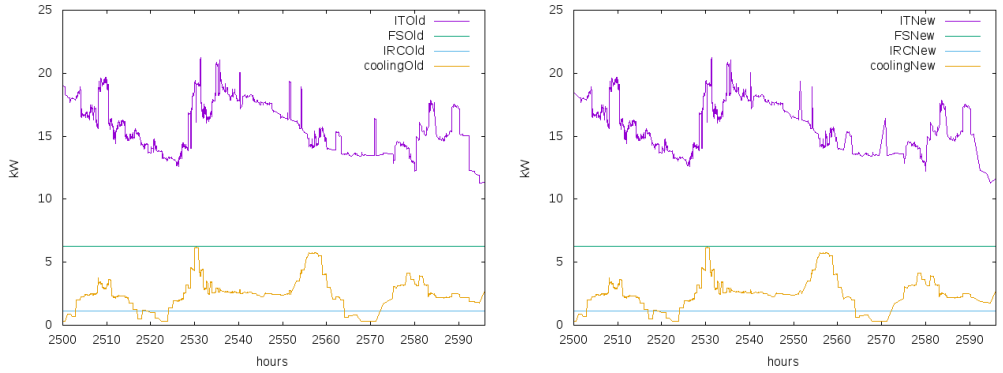

Figura 4.2.1.c: Consumo por componentes DC-Sim (izquierda) y SFIDE(derecha) en el escenario pequeño con temperaturas moderadas. En ambos modelos se obtienen valores iguales para todos los componentes: IT, FS, IRC y Cooling.

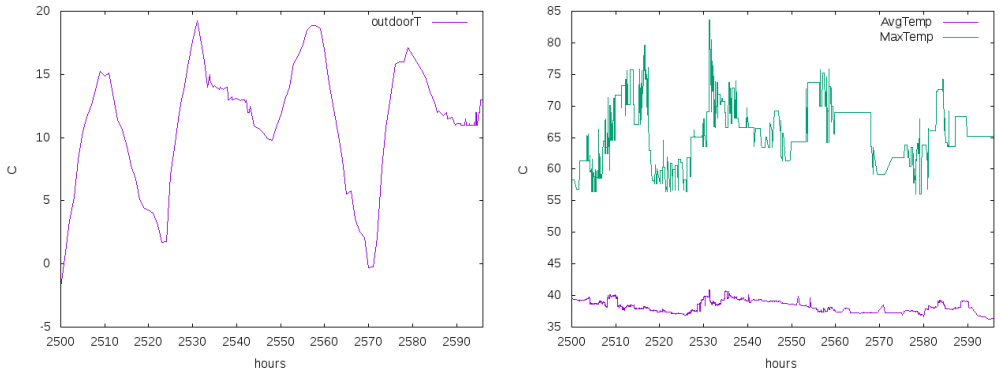

Figura 4.2.1.d: Temperatura exterior (izquierda) y de los servidores (derecha). La temperatura en los servidores está condicionada por la diferente carga de trabajo en cada instante.

Al incrementarse la temperatura exterior *(figura 4.2.1.d(izquierda))*, se puede apreciar en la *figura 4.2.1.c* que el consumo de *cooling* deja de ser constante, produciéndose "picos"de consumo que pueden llegar hasta los 6kW, coincidiendo con los máximos de temperatura durante el período, debido a la imposibilidad de realizar la refrigeración mediante *freeecooling*, siendo necesario el funcionamiento del *chiller*. Se observa en la *figura 4.2.1.d(derecha)* la correlación entre el consumo de *IT* y las temperaturas alcanzadas por los servidores.

- ***Temperatura alta***

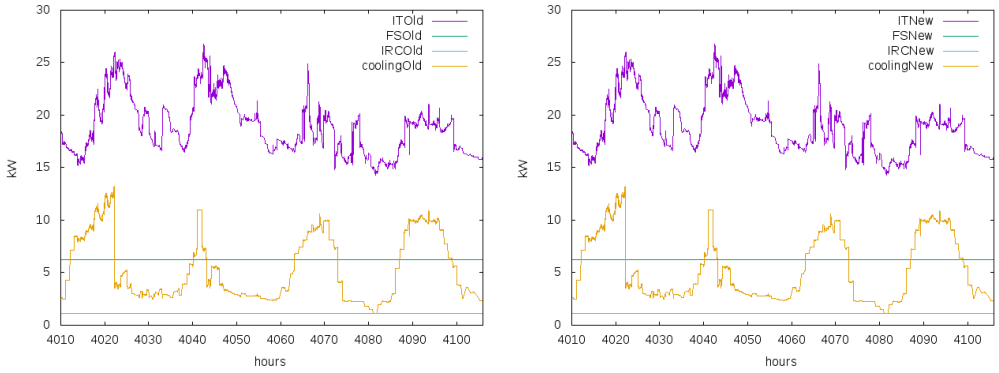

Figura 4.2.1.e: Consumo por componentes DC-Sim (izquierda) y SFIDE(derecha) en el escenario pequeño con temperaturas altas. En ambos modelos se obtienen valores iguales para todos los componentes: IT, FS, IRC y Cooling.

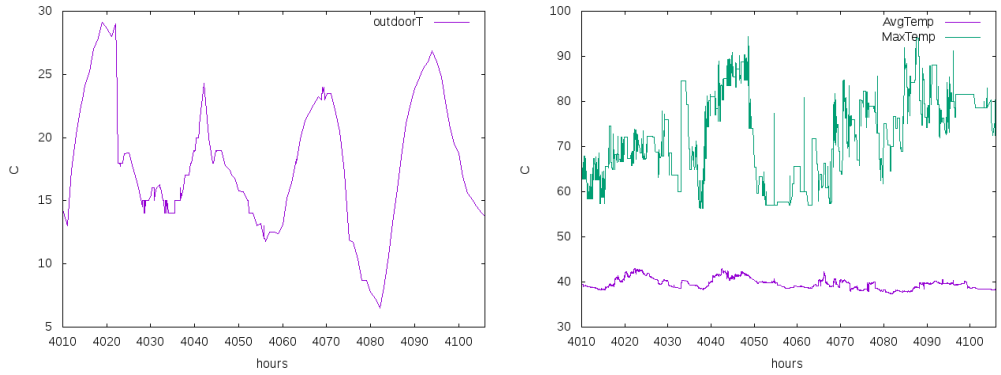

Figura 4.2.1.f: Temperatura exterior (izquierda) y de los servidores (derecha). La temperatura en los servidores está condicionada por la diferente carga de trabajo en cada instante.

El incremento en la temperatura exterior, reflejado en la *figura 4.2.1.f(izquierda)*, produce la consecuencia representada en la *figura 4.2.1.e*; el consumo de *cooling* se incrementa considerablemente al tener que intervenir de manera casi permanente el *chiller* en el proceso de refrigeración.

4.2.2. Resultados escenario mediano

- ***Temperatura fría***

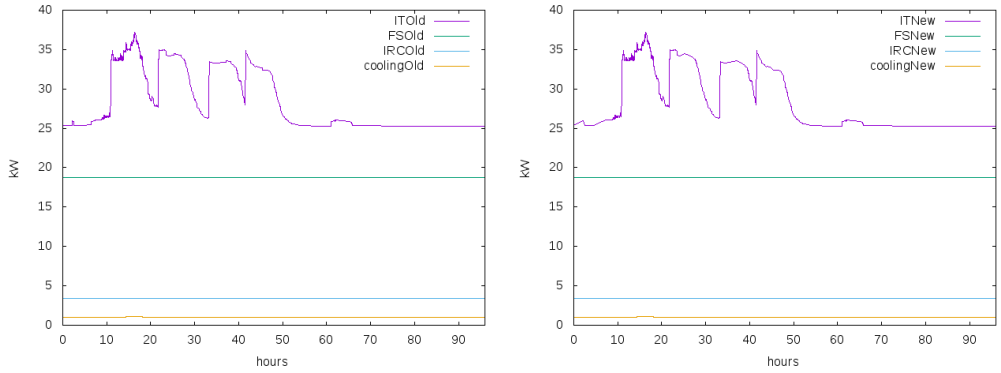

Figura 4.2.2.a: Consumo por componentes DC-Sim (izquierda) y SFIDE(derecha) en el escenario mediano con temperaturas frías. En ambos modelos se obtienen valores iguales para todos los componentes: IT, FS, IRC y Cooling.

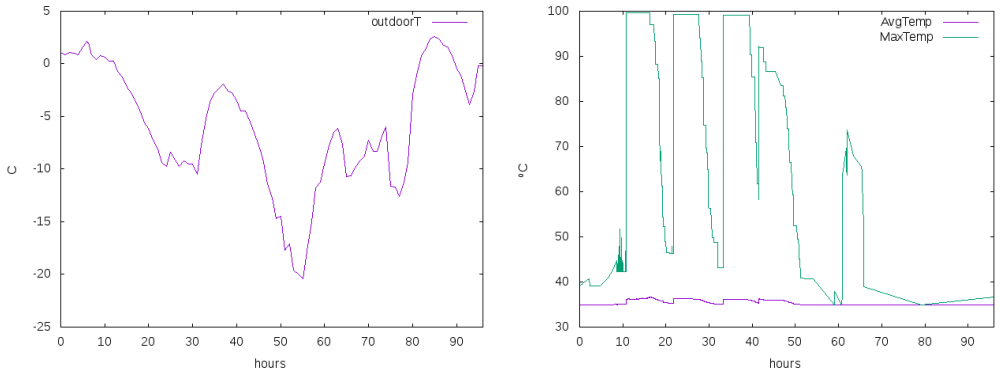

Figura 4.2.2.b: Temperatura exterior (izquierda) y de los servidores (derecha). La temperatura en los servidores está condicionada por la diferente carga de trabajo en cada instante.

Al cambiar de tamaño de escenario, lo primero que se observa en la *figura 4.2.2.a* es un incremento en los consumos de todos los componentes proporcional al incremento del número de servidores. En segundo lugar, teniendo en cuenta que el comportamiento del sistema de refrigeración es igual en ambos escenarios, se puede observar un leve aumento del consumo de *cooling*, debido principalmente al aumento de los galones de agua requeridos por los IRC y que será bastante más significativo en los escenarios posteriores. Además se sigue observando que los gráficos son iguales en ambos simuladores.

- ***Temperatura moderada***

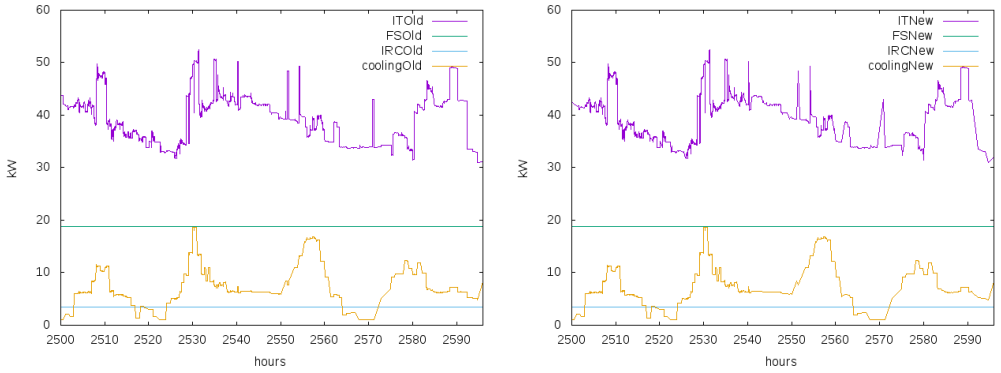

Figura 4.2.2.c: Consumo por componentes DC-Sim (izquierda) y SFIDE(derecha) en el escenario mediano con temperaturas moderadas. En ambos modelos se obtienen valores iguales para todos los componentes: IT, FS, IRC y Cooling.

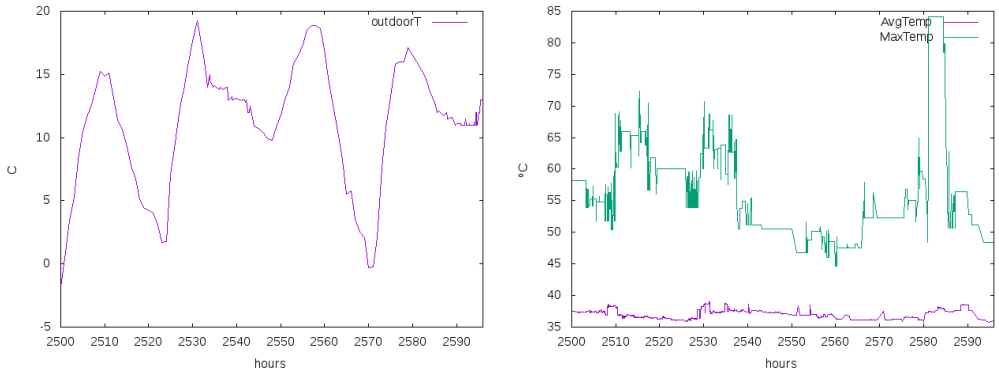

Figura 4.2.2.d: Temperatura exterior (izquierda) y de los servidores (derecha). La temperatura en los servidores está condicionada por la diferente carga de trabajo en cada instante.

El incremento de la temperatura exterior provoca el aumento del consumo de *cooling* de manera proporcional al producido en el escenario pequeño. La imposibilidad de refrigerar el sistema mediante *freecoling* en los momentos en que la temperatura exterior es más elevada, hace que el gráfico presente una serie de "picos"de consumo. El mayor número de servidores, además, hace que el flujo de agua necesario para la refrigeración de los IRC sea mayor, incrementando el consumo de la bomba.

- ***Temperatura alta***

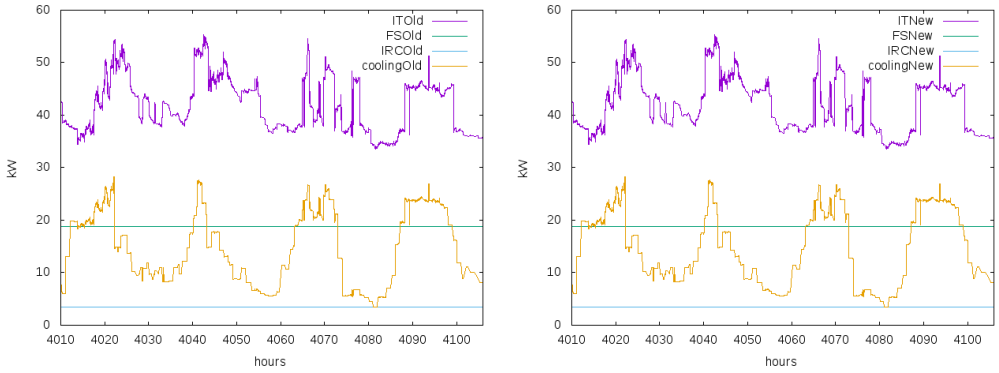

Figura 4.2.2.e: Consumo por componentes DC-Sim (izquierda) y SFIDE(derecha) en el escenario mediano con temperaturas altas. En ambos modelos se obtienen valores iguales para todos los componentes: IT, FS, IRC y Cooling.

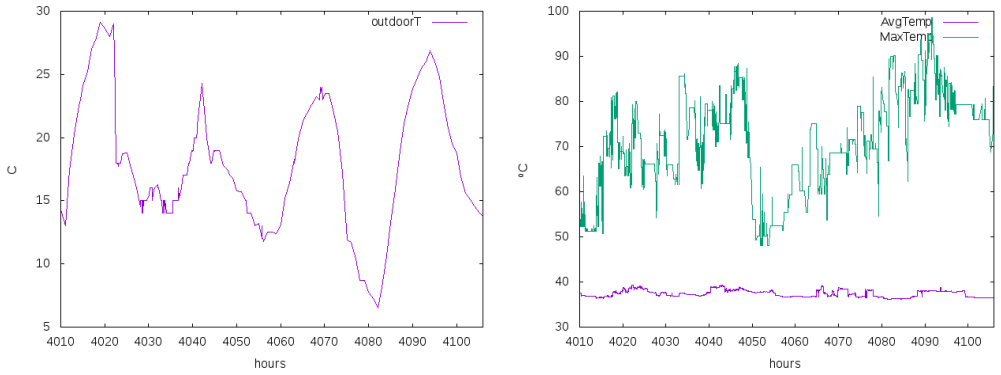

Figura 4.2.2.f: Temperatura exterior (izquierda) y de los servidores (derecha). La temperatura en los servidores está condicionada por la diferente carga de trabajo en cada instante.

En las figuras 4.2.2.e y 4.2.2.f se observa como el consumo de la refrigeración de los servidores y los ventiladores pueden llegar a superar al consumo de IT, disparando los consumos totales y las temperaturas en momentos de excesiva carga de trabajo en los servidores y en condiciones de alta temperatura exterior.

4.2.3. Análisis del tiempo de ejecución

Ambos escenarios se probaron sobra una máquina virtual de Ubuntu 16.04 con 3GB de RAM; el hardware utilizado fue: un Intel Core i5-5200U @ 2.20Ghz, 8 GB de RAM DDR3L y un SSD 128GB.

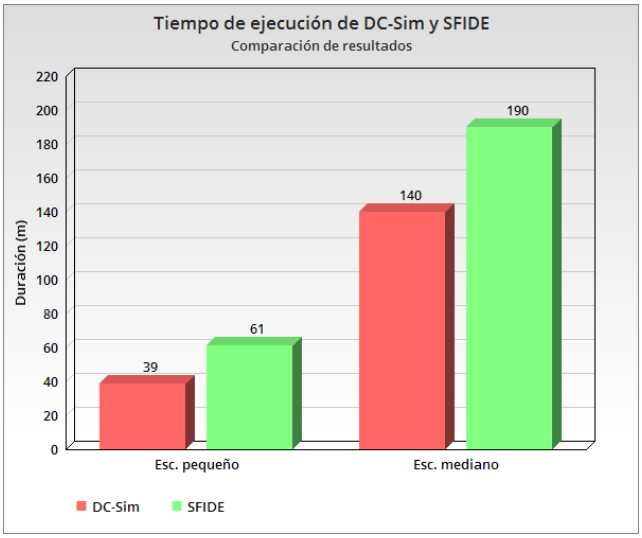

Figura 4.2.3:Tiempos de ejecución en DC-Sim y SFIDE

El tiempo de ejecución de ambos simuladores queda reflejado en la figura 4.2.3. Como puede observarse el tiempo de ejecución en SFIDE es mayor que en DC-Sim. El tiempo de ejecución de ambos simuladores escala con el número de trabajos y el número de servidores, pero en SFIDE

el hecho de tener que recorrer cada componente atómico en busca del menor sigma penaliza su tiempo de ejecución frente a DC-Sim. Además el uso de una memoria de estado sólido favorece los accesos aleatorios a los nuevos archivos de trabajo en DC-Sim.

Se conseguiría una mejora sustancial en el tiempo de ejecución de SFIDE pasando del simulador secuencial actual, a un simulador paralelo que permitiese la "división aproximada"del tiempo de ejecución entre el número de hilos para la búsqueda del menor sigma en el árbol de coordinadores y simuladores, y la ejecución de las funciones de transición.

4.3. Otras ejecuciones

Una vez validada la nueva implementación y con el fin de comprobar la flexibilidad y escalabilidad del modelo SFIDE, modificamos la configuración del data center de manera que cada rack tiene nueve servidores, a los que les asignamos una carga de trabajo equivalente a la utilizada en el escenario pequeño. Así, tenemos una configuración con 3 IRC's, 6 racks y 54 servidores, con un conjunto de 330.447 trabajos.

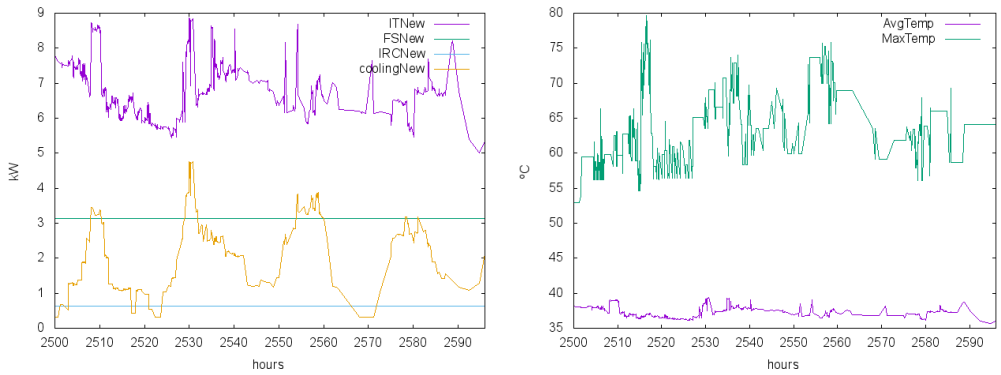

Figura 4.3.a: Consumo por componentes (izquierda) y temperatura de los servidores (derecha).

Como puede observarse en la figura 4.3.a, el resultado de la simulación refleja unos datos coherentes con respecto a los obtenidos anteriormente.

4.4. Conclusiones

El diseño e implementación del modelo SFIDE, objeto del presente trabajo, confirma la aplicabilidad del formalismo DEVS para el desarrollo de este tipo de aplicaciones. Con los resultados obtenidos se demuestra que la separación entre el modelo y el simulador es posible, permitiendo además una extraordinaria facilidad a la hora de añadir nuevas funciones en futuros desarrollos.

El hecho de que el modelo SFIDE sea independiente del simulador puede resultar interesante ya que, si se desea paralelizar los eventos para un tiempo determinado, e incluso el simulador de la aplicación, no es necesario cambiar el modelo. Se puede conseguir de esta manera una importante mejora en el tiempo de ejecución cuando el número de servidores sea muy elevado.

Para verificar la validez de los resultados obtenidos, se han ejecutado los mismos escenarios en el simulador DC-Sim y en SFIDE, comprobando que aquéllos son iguales para ambos en todas las situaciones.

La mayor flexibilidad y escalabilidad de SFIDE le convierte en una herramienta muy útil para la simulación de Data Center de diferentes topologías, con diferentes tipos de servidores e incluso en distintas localizaciones geográficas sin necesidad de realizar costosas adaptaciones. La estructura de comunicación entre los distintos componentes permite la incorporación de nuevos elementos de una manera inmediata.

Conclusions

The design and implementation of the SFIDE model, focus of the present study, confirms the applicability of the DEVS formalism for the development of this kind of applications. The results obtained show that the division between model and simulator not only is possible, but also leads to an extraordinary simplicity when adding new features in future developments.

The fact that the SFIDE model is independent from the simulator might be interesting, since, if the events (and even the application simulator) are to be parallelized for a certain time, it is not necessary to change the model. This way, a significant improvement in the execution time may be achieved, when the number of servers is very high.

To verify the validity of the results obtained, the same scenarios have been executed in the DC-Sim simulator and in SFIDE, and the results are the same in both of them and in any kind of situation.

SFIDE's great flexibility and scalability makes it a very useful tool for the Data Center simulation, with different topologies, different kinds of servers and even different geographic locations, and without the need for any expensive adjustments. The communication structure between the different components allows an immediate incorporation of new elements.

4.5. Futuros desarrollos

Este proyecto supone la base de nuevos desarrollos para la simulación de rendimiento en centros de datos integrados. En un futuro se crearán nuevos componentes para el simulador como la inclusión de nuevas políticas de refrigeración y/o un nuevo gestor de la temperatura de las CPU's.

Una de las posibles expansiones de este desarrollo sería añadir la posibilidad de emplear una política de minimización de consumo de refrigeración denominada *budget heat*. Para implementarla sería necesario un nuevo atómico, el cual recibiría por un puerto el sumatorio de los consumos dinámicos que lleguen a los IRC y un puerto del chiller que compruebe si el freecooling es posible con el calor y la temperatura actual.

En el momento en que reciba el nuevo consumo referente a cada trabajo debe ejecutar un bucle que consiga la porción del calor que esta extrayendo el Chiller se corresponde con cada IRC. Para saber la temperatura del chiller se debe extender el puerto que envía la temperatura del chiller al IRC para que llegue al nuevo atómico. Una vez hallados estos parámetros hay que probar todas

las posibles configuraciones de ventiladores y temperatura de inlet y marcar aquellas que no den problemas de temperaturas en las CPU's. Una vez conseguidas las configuraciones validas hay que discriminar aquellas que tenga el menor consumo posible y preservar dichas configuraciones en todos los servidores e IRC's.

En esta política de refrigeración sería interesante la inclusión de un simulador paralelo para DEVS, ya que todos servidores deben variar su estado y sus datos no son dependientes entre sí dando cabida al paralelismo previamente solicitado. Además de la mejora de rendimiento del simulador al recorrer el árbol de coordinadores en un tiempo proporcional al número de hilos de ejecución.

El modelo DEVS permitiría también el estudio de la variación de temperaturas. Sabemos que la temperatura de la CPU sigue una exponencial creciente, que afecta al consumo de *leakage* (P_{leak}). Actualmente sólo se calcula el estado estacionario, pero añadiendo una transición interna δ_{int} en el componente atómico Server se podrían calcular estos transitorios.

Otra de las opciones de ampliación de la funcionalidad es: crear un nuevo atómico planificador que, en vez de recibir el objetivo de las tareas desde un archivo de texto, elija el objetivo de los trabajos priorizando a aquel servidor que menos consumo produzca. Para ello sería necesario que todos servidores tengan un puerto que lleve al nuevo planificador, se realice una búsqueda del mínimo consumo y si se cumplen las restricciones enviar la tarea a ese servidor por el método actual.

Para facilitar el uso del simulador se podría crear una GUI, que permitiese la monitorización y gestión de los distintos componentes que conforman el data center.

Bibliografía

[1] MARINA ZAPATER, ATA TURK, JOSE M. MOYA, JOSE L. AYALA AND AYSE K. COSKUN, *Dynamic Workload and Cooling Management in High-Efficiency Data Centers*

[2] MARINA ZAPATER, OZAN TUNCER, JOSE L. AYALA, JOSE M. MOYA, KALYAN VAIDYANATHAN, KENNY GROSS, AND AYSE K. COSKUN, *Leakage-Aware Cooling Management for Improving Server Energy Efficiency*

[3] I. PENAS, M. ZAPATER, J.L. RISCO, J.L. AYALA AND AYSE K. COSKUN, *Sfide: A simulation infrastructure for Data Centers*

[4] JOSÉ L. RISCO AND SAURABH MITAL, *"xDEVS r20150903"*

[5] JOSÉ L. RISCO AND SAURABH MITAL, *Netcentric System of Systems Engineering with DEVS Unified Process*

[6] JOSÉ DAMIÁN FERRER QUINTANA, *Centro de Proceso de Datos: el cerebro de nuestra sociedad*

[7] ALFONSO URQUÍA, *Modelado de sistemas mediante DEVS*

Agradecimientos

A Jose Luis Ayala, por darnos la posibilidad de afrontar el reto que nos ha supuesto la realización de este proyecto.

A José Luis Risco, Marina Zapater e Ignacio Penas, por su inestimable ayuda para la su elaboración.

Autorización de difusión

Autorización para la difusión del Trabajo Fin de Grado y su depósito en el Repositorio Institucional E-Prints Complutense

Los abajo firmantes, alumno y tutor del Trabajo Fin de Grado (TFG) en Ingeniería de Computadores de la Facultad de Informática, autorizan a la Universidad Complutense de Madrid (UCM) a difundir y utilizar con fines académicos, no comerciales y mencionando expresamente a su autor, el Trabajo Fin de Grado (TFG) cuyos datos se detallan a continuación. Así mismo autorizan a la Universidad Complutense de Madrid a que sea depositado en acceso abierto en el repositorio institucional con el objeto de incrementar la difusión, uso e impacto del TFG en Internet y garantizar su preservación y acceso a largo plazo.

TÍTULO del TFG: **Implementación de un simulador de rendimiento, consumo y coste para Centros de Datos integrados en un Smart Grid**.

Curso académico: 2016/2017

Nombre de los Alumnos: **Fernando Baraibar Lópezy Roberto Montero Cobo de Guzmán**.

Tutores del TFG: **José Luis Ayala Rodrigoy Marina Zapater Sancho**, Departamento de Arquitectura de Computadores y Automática.

Fdo.: José Luis Ayala Rodrigo

Fdo.: Marina Zapater Sancho

Fdo.: Fernando Baraibar López

Fdo.: Roberto Montero Cobo de Guzmán