Proyecto de Trabajo Fin de Carrera:

PARALELIZACIÓN DE LA EJECUCIÓN EXTENSIVA DE EJEMPLOS DE TEST PARA HERRAMIENTAS DE PLANIFICACIÓN DE SISTEMAS DE TIEMPO REAL

Javier Barba Rueda
Directores: J. Carlos Palencia Gutiérrez
y J. Javier Gutiérrez García

UNIVERSIDAD DE CANTABRIA

Licenciatura en Física
Facultad de Ciencias, Universidad de Cantabria
Marzo 2011

Índice

Capítulo 1
Introducción

1.1.- Sistemas de Tiempo Real

Un sistema de tiempo real es una combinación de un computador, los dispositivos de entrada y salida de éste y el *software*, en donde además:

- Hay una gran interacción con el entorno que lo rodea;
- Este entorno cambia con el tiempo;
- El sistema controla y reacciona frente a distintos aspectos del entorno;

Como resultado de ello, los requisitos de tiempo se imponen en el *software*. Éste además debe ser concurrente, es decir, que sólo se puede ejecutar un proceso en un instante exacto.

Para estudiar y analizar estos sistemas de tiempo real se utilizan muchas veces herramientas *software* que analizan a gran velocidad grandes cantidades de datos. Una de estas herramientas es MAST.

1.1.1.- MAST

MAST (*Modeling and Analysis Suite for Real-Time Applications*) es una colección de herramientas *software* de código abierto (*open source*) creada por el grupo de Computadores y Tiempo Real (CTR) de la Universidad de Cantabria para modelar aplicaciones de tiempo real y estudiarlas posteriormente.

La actual versión y más reciente de MAST es la 1.3.8.0 para Windows y Unix que se puede descargar desde la página de dicha herramienta (http://mast.unican.es/) y ha sido usada en este proyecto.

El *software* incluye varias herramientas programadas en lenguaje ADA. Estas herramientas son:

- **gmast_analysis** – Herramienta con una interfaz gráfica para el análisis de los sistemas de tiempo real.
- **gmasteditor** – Herramienta con una interfaz gráfica para crear y editar los modelos MAST.
- **gmastresults** – Herramienta con una interfaz gráfica para visualizar los ficheros de salida del análisis de los modelos MAST (*outputs*).
- **mast_analysis** – Herramienta principal de análisis y cálculo de los sistemas de tiempo real. Su ejecución se realiza a través del interfaz gráfico gmast_analysis o a través de MS-DOS en Windows o la *shell* de Unix.
- **mast_xml_convert** – Herramienta que exporta los resultados obtenidos en los ficheros de entrada (modelos MAST o *inputs*) a ficheros de formato de editores de texto para tener una visualización más clara de dichos modelos MAST.
- **mast_xml_convert_results** - Herramienta que exporta los resultados obtenidos en los

ficheros de salida (*outputs*) a ficheros de formato de editores de texto para tener una visualización más clara de los resultados obtenidos.

Este informe se centra más la herramienta "mast_analysis" dado que será la herramienta principal que se ejecutará en Calderón para analizar distintos modelos.

1.1.2.- Estructura del funcionamiento de MAST

La edición de estos modelos MAST (o ficheros *inputs* de mast_analysis) define totalmente el sistema de tiempo real que será analizado.

Para definir estos sistemas se describen el número, tipo y la organización de sus elementos de tiempo real. Estos elementos, para un modelo MAST, son:

- **Procesador** (*Processor*): Elemento *hardware* que interpreta las instrucciones y los datos de los procesos en la computadora;
- **Planificador** (*Scheduler*): Es la entidad *software* que controla al procesador. Esta entidad está asociada a un procesador y define los rangos y tipos de prioridades con los que trabaja;
- **Operación** (*Operation*): Este elemento equivaldría al ejecutable o al código de un programa que genera un proceso. Estas operaciones poseen un tiempo máximo de ejecución (peor tiempo de respuesta) definido;
- **Tarea** (*Scheduling server*): las tareas son los *threads* o procesos generados por una determinada operación. Estas tareas son interpretadas en un procesador y gestionadas por el planificador de éste. Cada tarea es inherente a una prioridad dentro del planificador, lo que determina el orden de estas tareas al ser ejecutadas por un procesador;
- **Transacción** (*Transaction*): las transacciones son concatenaciones de tareas. Estas tareas son ejecutadas unas detrás de otras siguiendo un determinado orden y unas características temporales, tales como el periodo con que debe ejecutarse la serie de tareas, o el plazo de tiempo (*deadline*) en que debe finalizarse. Aunque a veces estas características temporales no están definidas por periodos, sino por otras directrices temporales en las que aquí no se hará hincapié aquí. En las transacciones se relacionan las operaciones y las tareas en un elemento que se llama *activity*;
- **Red** (*Network*): La red es en esencia un procesador, con la diferencia de que este elemento en concreto distribuye la información que se comparte entre distintos procesadores en un sistema distribuido (de varios procesadores, no sólo uno), por lo que posee ciertos atributos que lo diferencian de un procesador normal.

En una transacción se pueden encontrar tareas de distintos procesadores interconectadas entre ellas. Para ello se usa una red que las una.

Puede darse el caso de que una determinada transacción termine toda la serie de tareas más tarde de lo que se indica en el plazo. En este caso la transacción no es planificable (*schedulable*), es decir, que no se ha podido resolver dentro de los límites de tiempo establecidos. Si todas las transacciones son planificables, entonces el sistema también será planificable. Sin embargo, una sola transacción no planificable implicaría que el sistema tampoco lo es. Esto es lo principal a la hora de analizar un sistema de tiempo real, aunque el análisis puede conllevar otra serie de cálculos y

optimizaciones como los que se verán más adelante.

La estructura que forman los elementos de los modelos MAST, para un sólo procesador es:

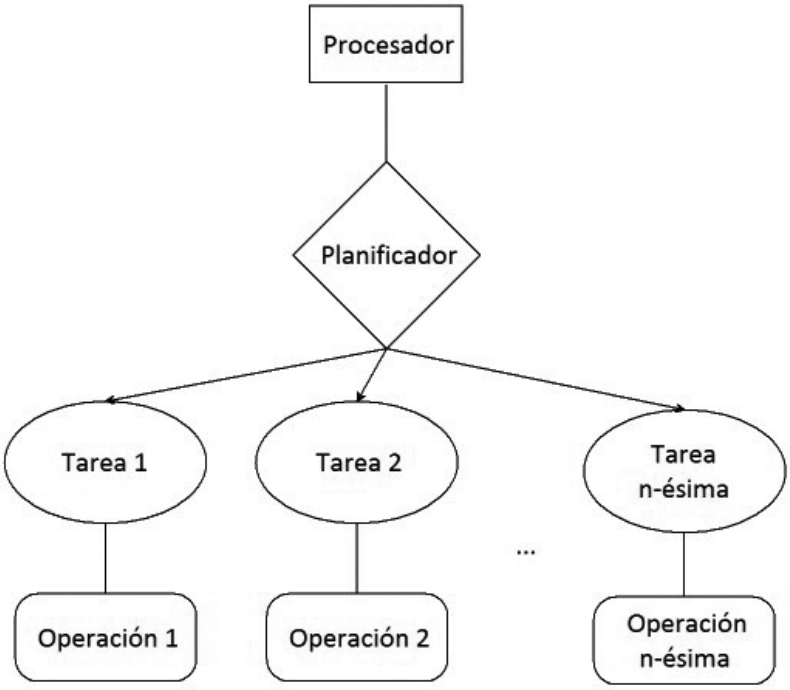


Figura 1. Diagrama de la organización de elementos en MAST para un sólo procesador.

El planificador está asociado a un procesador. Las tareas están contenidas en el planificador que las gestiona, y a su vez estas tareas están asociadas a una operación.

En el caso de que existan más procesadores, cada uno poseerá sus propias tareas, pero no interaccionarán con los elementos de otro procesador hasta llegar a una transacción.

A continuación se representa un esquema del funcionamiento de una transacción:

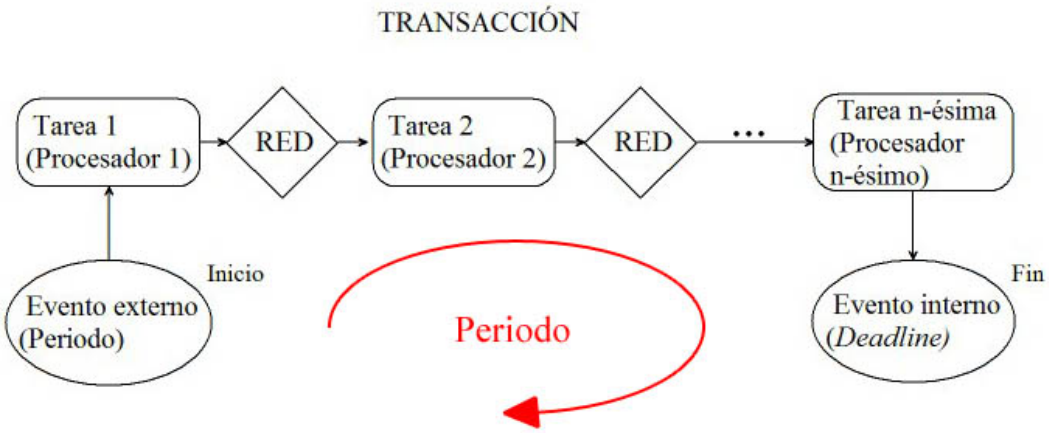


Figura 2. Diagrama del funcionamiento de una transacción con tareas de distintos procesadores enlazadas por medio de la red.

En cada transacción, al iniciarse ésta, comienza una serie de ejecuciones de tareas, en el caso de la figura 2 cada una de un procesador distinto, enlazadas por medio de mensajes de la red. Una vez finalizado se comprueba que desde el inicio hasta el final se ha tardado menos tiempo del permitido por el *deadline* de la transacción. Una vez iniciada la transacción, ésta se iniciará de nuevo cuando un tiempo igual al periodo se cumpla.

1.1.3.- Cómo ejecutar mast_analysis

Para ejecutar el archivo binario de análisis de modelos MAST (mast_analysis), es necesario usar en el caso de trabajar con un sistema operativo Windows, una ventana de Símbolo del Sistema (antiguamente MS-DOS), en el caso de un sistema operativo de Unix, una *shell*. En este proyecto se ha trabajado casi exclusivamente en un sistema operativo de Unix, concretamente se ha usado Ubuntu 9.10.

Una vez dicha ventana esté abierta, se ha de usar el siguiente comando para usar la herramienta:

```
./mast_analysis herramienta [opciones] input [output]
```

Comando 1. Forma de ejecutar mast_analysis en la *shell* de Unix o en MS-DOS.

La herramienta elegida puede variar enormemente los resultados obtenidos y el tiempo de ejecución, ya que los algoritmos usados son distintos para cada herramienta. Las herramientas usadas se explicarán en el siguiente apartado.

Las opciones no son estrictamente necesarias, pero pueden aportar alguna información relevante sobre el sistema, además del simple análisis sobre el sistema y su planificabilidad.

El fichero de entrada (*input*) al igual que el uso de una herramienta es obligatorio. El *input* es el fichero que define el modelo MAST que será analizado.

El fichero de salida (*output*) es opcional, al igual que el uso de alguna opción. Estos ficheros de salida son archivos en los que se guarda cierta información del sistema. Si no se usa un *output*, se imprimirá por pantalla (en la misma *shell*) dicha información.

1.1.4.- Las herramientas de MAST

Como se ha comentado anteriormente, existen distintas formas de analizar un mismo sistema de tiempo real, ya que existen distintos algoritmos. A continuación se explican cuáles son los 3 más comunes y que se usarán en este proyecto.

Hay que tener en cuenta que ninguna de las tres herramientas resuelve exactamente los sistemas que son analizados. Es prácticamente imposible analizar completamente y con total exactitud un sistema si éste no es demasiado simple. Por ello, se toman ciertas aproximaciones pesimistas para resolverlo de manera más sencilla.

La primera herramienta, ***holistic***, es a priori la más simple de las tres que se van a explicar. Toma ciertas suposiciones y resuelve de manera cercana a la realidad dicho sistema. La suposición más

relevante de este algoritmo implica que las tareas de una misma transacción son independientes entre ellas, lo que simplifica mucho el cálculo, convirtiendo a esta herramienta a su vez en la menos exacta pero la más veloz. Se usó esta herramienta en algunos análisis no contemplados para esta memoria a la hora de adquirir experiencia con MAST.

La segunda y la tercera herramienta, ***offset based*** y ***offset based optimized***, provienen de una misma estrategia de cálculo, afinando más las suposiciones dadas en el caso *holistic*, ya que "mejora" el pesimismo del análisis, tomando esta vez la suposición de que las tareas de cada transacción no son independientes (lo que complica el análisis pero lo hace más acertado). La versión *optimized* evidentemente está más afinada en sus cálculos, analizando las relaciones más inmediatas de las tareas de una transacción, lo que se traduce en más cálculos pero también en un resultado más cercano al correcto. Se usaron estas dos herramientas para las ejecuciones que se verán más adelante.

Por otro lado, a estas herramientas de análisis hay que sumar otras herramientas distintas que se llamarán de ahora en adelante "opciones de ejecución" (las opciones vistas en el apartado 1.1.3).

Estas opciones se añadirán a los análisis realizados, calculando nueva información del sistema, información adicional que consumirá más tiempos de cálculo que un simple análisis del sistema para comprobar la planificabilidad del mismo.

En este proyecto las opciones principales serán las de reasignación de prioridades, aunque se podría hablar de otras herramientas como el ***slack***, una herramienta que indica el porcentaje que deben variar los tiempos de peor caso de las operaciones para que éstos sean máximos y el sistema siga siendo planificable.

Pero se centrará el interés en la reasignación de las prioridades. Esto significa que MAST analizará un sistema varias veces si éste resulta no ser planificable. En cada nuevo análisis MAST variará las prioridades de las operaciones en busca de una nueva disposición de éstas, de manera que de este modo el sistema sí sea planificable. La principal herramienta para llevar a cabo esto es la denominada *HOPA*, aunque existe otra herramienta, menos eficiente, llamada *Annealing*.

Annealing reasigna las prioridades de manera aleatoria.

HOPA (ver la referencia **[7]** de la bibliografía) sin embargo no lo deja al azar, el algoritmo busca mínimos locales en determinadas funciones del sistema para trabajar en ellos. El número máximo de iteraciones que puede realizar *HOPA* para encontrar una solución es configurable. En este proyecto se configurará para que realice un máximo de 180 iteraciones en el análisis, o hasta que el sistema sí sea planificable, por lo que es normal que el tiempo de estos análisis no planificables sea considerablemente mayor a uno planificable desde el principio.

En ocasiones, es normal ver que MAST indica que un sistema para una determinada carga de trabajo no sea planificable, y al aumentar dicha carga (más tarde se medirá esa carga de trabajo con el nombre de utilización) el sistema sí sea planificable, cuando a priori se podría pensar que no tiene sentido. Pero hay que tener en cuenta que como el análisis no es exacto, puede ocurrir que

MAST no encuentre un resultado óptimo para un caso pero sí para el siguiente de más carga.

1.2.- Supercomputación

El grupo de Arquitectura y Tecnología de Computadores (ATC) de la Universidad de Cantabria dispone de un *cluster* de computadores llamado Calderón.

El término *cluster* es definido por **[5]** como "*un sistema de procesamiento paralelo o distribuido, que consiste en una colección de computadoras interconectadas que trabajan como un sólo recurso computacional*".

Como se explicará más adelante, en los objetivos de este proyecto, es necesario el uso de un *cluster* para poder paralelizar el *software* de análisis de sistemas de tiempo real.

Este clúster de cálculo paralelo se compone de los siguientes elementos:

- 1 Front-end o servidor HP Proliant DL 380 3º Generación bi-procesador.
- * 15 nodos HP Proliant DL 145 1º Generación bi-procesador con AMD Opteron 248.
- 20 nodos HP Proliant DL 145 2º Generación bi-procesador con AMD Opteron 265 dual core.
- 6 nodos HP Proliant DL 145 2º Generación bi-procesador con AMD Opteron 275 dual core.
- 30 nodos HP Proliant DL 160 5ª Generación bi-procesador con INTEL Xeon 5472 quad core.
- 1 "enclosure" HP DL1000: 4 nodos HP Proliant DL 170h 6ª Generación bi-procesador con INTEL Xeon X5550 quad core.
- ** 5 "enclosure" SUPERMICRO SuperServer 6026TT-HT: 20 nodos six-procesador INTEL Xeon X5650 six-core.
- 1 nodo Supermicro SYS-6016GT-TF-TC2 con INTEL Xeon 5504 quad core y 2 tarjetas gráficas Nvidia Tesla C1060 GPU Cards.
- 3 nodos Sun UltraSPRAC T1 y T5120 (T2), con procesadores Niagará.
- 6 Switch GigaBit Ethernet (10/100/1000) de 24 puertos cada uno para gestión.
- 1 Switch Myrinet bi-canal a 2 Gbps por canal para comunicación paralela con 16 puertos.
- 1 Switch Infiniband bi-canal a 10 Gbps por canal para comunicación paralela con 24 puertos.
- 1 Sistema de gestión de consolas KVM + monitor TFT.
- 1 Switch GigaBit Ethernet (10/100/1000) de 48 puertos para gestión IPMI (consola).
- Sistema Operativo Linux Debian.

* Dado de baja en el *cluster* en febrero del 2011.
** Añadido al *cluster* en febrero del 2011.

Ello hace un total de 570 procesadores en el *cluster*, con una memoria de almacenamiento de aproximadamente 24.5 TB y una memoria RAM total de 1655 GB. Hasta el mes de febrero, mes en que se añadió una gran cantidad de procesadores y más memoria, estos números eran algo inferiores, con 430 procesadores, 19.5 TB de disco duro y 881 GB de memoria RAM.

El *cluster* se encuentra enclaustrado en un Centro de Procesamiento de Datos (CPD) llamado CUDO

InfraestruXure de APC, compuesto de 16 *racks* o bastidores, con 4 unidades de refrigeración que mantienen la temperatura del CPD entre 20 y 25ºC y una humedad relativa del 50% para optimizar el trabajo del *cluster* y mantenerlo en perfectas condiciones de seguridad.

Imagen 1. El CPD CUDO InfraestruXure de APC que enclaustra el *cluster* (además de algunos otros equipos de gestión informática de la Universidad de Cantabria). El CPD dispone de 30 *racks* (de los que Calderón ocupa 16) a los que se accede por el pasillo que atraviesa el CPD (cuya puerta se ve en la imagen) o por la parte trasera de los *racks*. En la imagen se puede ver al administrador del *cluster* (José Ángel Herrero Velasco) accediendo al *cluster* de forma local.

Además, en caso de caídas de tensión u otros problemas externos, cada *rack* dispone de su propia batería (ver imagen 3) para alimentarse, como medida de emergencia. Además, se dispone de un robot (HP MSL6060) con una capacidad total de 60 TB de almacenaje, administrado mediante un sistema *software* de gestión de copias HP open View Satore Works DATA PROTECTOR 6.0, para crear copias de seguridad de los trabajos en caso de problemas, además de un completo sistema de alertas y medidas de contingencia automatizadas.

Como ya se ha explicado, el *cluster* dispone de distintos procesadores y nodos (unidades independientes de procesamiento). Estos nodos están agrupados de distintas maneras, es decir, dependiendo el tipo de uso que se les quiera dar; hay creados distintos grupos, para compilación, ejecución de trabajos secuenciales, o para ejecución de trabajos paralelos. Se puede ver la agrupación de dichos nodos en el Anexo I.

Hay que añadir que existe una diferencia entre paralelizar un programa y ejecutar un trabajo paralelo. Paralelizar un programa implica ejecutar de forma simultánea en distintos procesadores distintas copias de un mismo programa, con distintos atributos o valores de entrada. Por otro lado, ejecutar un trabajo paralelo puede ser llevado a cabo por varios procesadores que irán resolviendo de manera independiente cada proceso de éste e interaccionando entre ellos de vez en cuando, siendo todo ello una estrategia para ejecutarlo de manera más rápida. No todos los programas están programados para poder ser trabajos paralelos, como es el caso de MAST, por lo que cuando se hable de paralelizar a partir de ahora querrá decir que se ejecutarán procesos de MAST distintos

en procesadores independientes, con diferentes datos de entrada.

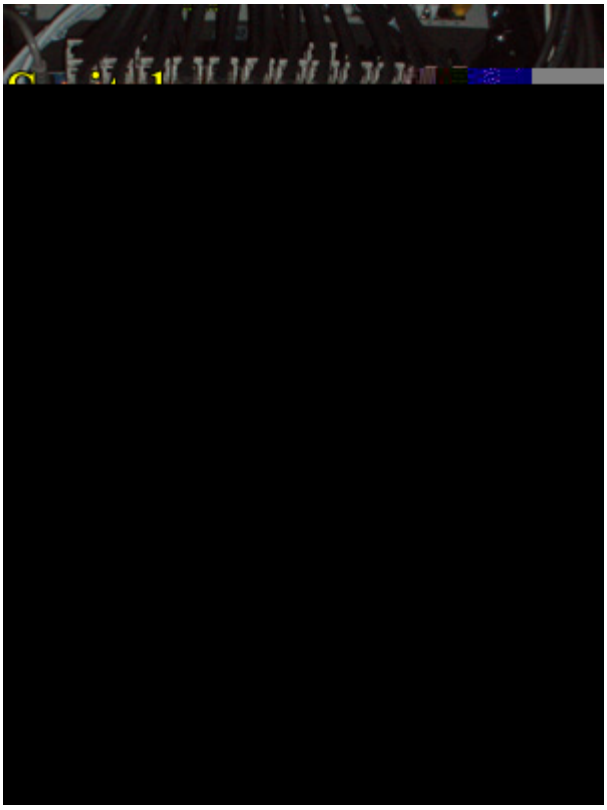


Imagen 2. Fotografía de unos nodos de cálculo en un *rack* del CPD. Se pueden observar los diferentes nodos iguales y apilados unos encima de otros. Cada uno de los nodos se conecta directamente con el switch que se encuentra sobre todos ellos. Este *switch* se encarga de gestionar las señales de entrada o de salida de los nodos con los potentes routers que utiliza el *cluster* para relacionarse con el exterior.

Imagen 3. Fotografía de las baterías de emergencia que alimentan todos los nodos de cálculo y el switch de cada *rack*. Estas baterías se localizan en el *rack*, bajo los nodos de cálculo (ver imagen 2). Se usan en caso de corte eléctrico para minimizar las posibles pérdidas.

1.2.1.- Ventajas del uso de un *cluster*

Hasta hace pocos años, disponer de un supercomputador capaz de resolver complicados problemas computacionales suponía un desembolso considerable para la comunidad científica. Además surgía un problema cuando se creaba una cola larga de espera para poder usar dicho supercomputador por varios usuarios, cada uno de ellos limitado a una cantidad máxima de tiempo de ejecución.

Con el paso del tiempo, los computadores personales se hicieron más y más populares, dado su

bajo coste, cualquier científico podía permitirse económicamente adquirir uno para su propio e ininterrumpido uso. Pero dado que estos computadores personales eran muy inferiores en prestaciones y potencia, existía la necesidad de encontrar un término medio.

En este punto nace la idea de conectar varios computadores intermedios a una red de altas prestaciones, en las que se puede ejecutar de manera paralela e independiente distintos programas. Con los debidos cuidados se puede mantener a dicho conglomerado de computadores interconectados trabajando las 24 horas del día todos los días del año. Esto presentaba muchas de las ventajas que se deseaban: uso simultáneo del recurso computacional, posibilidad de ejecución en una máquina que no se quede "inservible" mientras trabaja durante un largo periodo de tiempo, posibilidad de paralelización de un programa con la consecuente reducción de tiempo de ejecución del mismo, etc.

El uso simultáneo del recurso computacional implica poder usar el mismo computador (o *cluster* de computadores) por varios usuarios simultáneamente sin que dichos procesos interactúen entre ellos (retrasándose y usando recursos del sistema que obstaculicen la ejecución de otro proceso independiente).

La posibilidad de ejecución de un programa en una máquina, sin que ésta quede bloqueada hasta que finalice dicha ejecución, tiene que ver con el punto anterior. Al poder ejecutar distintos procesos de manera independiente, se puede usar el *cluster* pese a que éste esté siendo saturado de trabajo (Calderón gestiona un sistema de colas de trabajos, dejando en lista de espera trabajos que no puedan ser atendidos hasta que existan recursos para ello). En un computador personal, al ejecutar un programa, éste puede requerir ciertos recursos del sistema que impidan usar dicho computador para otras labores, además de que este proceso podría interactuar con otros procesos, por ejemplo los del mismo sistema operativo, u otros procesos de fondo, y ser retrasado innecesariamente pudiendo estropear mediciones de tiempo.

Por otro lado existe la posibilidad de ejecutar un trabajo paralelo, como ya se ha comentado, ejecutando un único programa que toma varios procesadores dividiendo su arquitectura interna en todos ellos. Este no es el caso de MAST, aunque sí puede ser paralelizado, ejecutando simultáneamente varias copias de éste en distintos procesadores, con lo que se pueden realizar muchos análisis y obtener grandes cantidades de información de manera rápida y cómoda.

También existe la ventaja de cargar al *cluster* con un trabajo y poder visualizar en el *cluster* cómo está resultando dicha ejecución mientras éste está trabajando sin interferir en dicha ejecución. Pudiendo ser avisados (en el caso de Calderón al menos) por correo electrónico de las fases de ejecución (inicio del procesado, fin, interrupción, suspensión de la ejecución, etc).

Como resumen, con un *cluster* se puede ejecutar durante grandes periodos de tiempo muchos programas pesados sin la necesidad de dejar un computador encendido e inutilizable en un despacho. Siendo dicho computador accesible a varios usuarios distintos sin problemas de interacción entre programas y procesos.

1.3.- Objetivos

El objetivo de este trabajo es implementar los mecanismos necesarios que permitan realizar sobre Calderón la ejecución en paralelo de análisis y optimización de sistemas con las herramientas de MAST. Para ello, se desplegará el trabajo sobre los procesadores que haya disponibles y se recopilarán los resultados al final de la ejecución.

Disponer de una herramienta que permita la ejecución en paralelo de las herramientas de MAST es muy importante dentro del trabajo de investigación que se desarrolla en el grupo CTR. Cuando se diseña un nuevo algoritmo de análisis de planificabilidad o de asignación de parámetros de planificación, además de la demostración teórica se requiere una validación experimental. Esta validación se realiza normalmente aplicando los algoritmos desarrollados sobre miles de ejemplos simulados que requieren una gran cantidad de cómputo. Por ejemplo, en el trabajo **[6]** se muestran los resultados de la aplicación de un nuevo algoritmo sobre un conjunto de sistemas de test de referencia; el número de ejemplos de test ejecutados para obtener esos resultados fue de 110000, y este número es una parte muy pequeña de los test totales realizados. Así pues, la ejecución de estos ejemplos en paralelo puede suponer una reducción considerable en el tiempo de validación de los algoritmos.

Dentro del objetivo general del proyecto, se van a desarrollar los siguientes objetivos específicos:

- Estudio de los mecanismos de ejecución en paralelo de Calderón y realización de ejemplos sencillos.
- Estudio de las herramientas disponibles en MAST y experimentación con ejemplos sencillos.
- Desarrollo de una herramienta que permita automatizar la ejecución en paralelo sobre Calderón de diferentes instancias de MAST con diferentes sistemas de entrada. Esta herramienta es la parte central de este trabajo y consistirá en el desarrollo de una serie de *scripts* de Calderón, que usados conjuntamente permitirán la ejecución en paralelo baterías de análisis de sistemas con MAST y la obtención de sus resultados.
- Realización de una herramienta de generación automática de sistemas de entrada a MAST, para la obtención de los ejemplos necesarios para las pruebas de ejecuciones paralelas. Estos ejemplos consistirán en sistemas a analizar con diferentes cargas de trabajo (variables del 1% al 99% de uso de los recursos).
- Evaluación de posibles beneficios que se obtienen en la ejecución en paralelo de los problemas típicos que tiene que resolver MAST. Esta evaluación consistirá en la estimación de los tiempos necesarios para analizar los sistemas típicos con distintas cargas que se generen con la herramienta propuesta en el punto anterior.
- Evaluación del rendimiento que presentan los diferentes grupos de procesadores de Calderón para ejecutar los algoritmos de análisis y optimización de MAST.

Otro objetivo inherente de un proyecto fin de carrera es la investigación que debe realizar el alumno y su adquisición de experiencia, en este caso en materia de tiempo real, el uso de *cluster*s y otros de carácter secundario como son la paralelización de software, el uso de Unix o la programación de herramientas para este proyecto.

1.4.- Estructura y organización de la memoria.

En esta memoria se presentan unos objetivos después de haberse explicado los medios de que se dispone (como herramienta de tiempo real y el *cluster* Calderón).

Una vez expuestos los objetivos se procederá a explicar cómo debe ser usado el *cluster* Calderón, ya que al ser una herramienta compleja y usada por muchos usuarios, es conveniente seguir unas directrices para su buen uso. Además de que también es necesario conocer cómo funciona a nivel práctico este *cluster*. Éste será el capítulo 2.

A continuación, en el capítulo 3 se explicarán las herramientas básicas creadas para este proyecto. Entre ellas se encuentran generadores de modelos de MAST, de *scripts* de ejecución de Calderón, lectores de grandes cantidades de archivos de salida de MAST, etc. Todos ellos cumplen un cometido distinto en la sistematización del proceso de paralelizar varias copias de MAST. También se detalla la herramienta más importante, que usando la mayoría de las explicadas anteriormente crea una serie de *scripts* y de modelos MAST, además de los *scripts* necesarios para gestionar tal cantidad de archivos generados.

A continuación se explicará en el capítulo 4 el procedimiento seguido para paralelizar los trabajos, así como el análisis de dicha paralelización y los resultados obtenidos. También se estudiarán algunas características de Calderón, como las diferencias de sus grupos de procesadores (ver el Anexo I).

El capítulo 5 relatará las conclusiones derivadas del capítulo 4 y de la experiencia en general paralelizando la herramienta de tiempo real en el *cluster*.

Por último se entrega la bibliografía de este proyecto, los agradecimientos del autor y dos anexos, el primero sobre los grupos de procesadores de Calderón, y el segundo sobre las herramientas programadas para este proyecto.

Capítulo 2
Manejo de Calderón

2.1.- Funcionamiento del *cluster* Calderón

El *cluster* es usado a día de hoy por cerca de una treintena de usuarios, que en muchas ocasiones coinciden ejecutando varios trabajos a la vez. Para poder organizar dichos trabajos de manera ordenada, que no interfieran entre ellos, o incluso para que no entorpezcan el rendimiento del *cluster*, han de respetarse ciertas normas de ejecución y uso.

Para poder ejecutar los trabajos, ha de elaborarse un *script* con la información de ejecución necesaria para ser puesto en la cola de prioridades y trabajos que gestiona Calderón. Éste necesita conocer ciertos datos de la ejecución previa a su puesta en marcha. Algunos de dichos datos son las rutas de trabajo, datos de los archivos de salida y error, duración estimada de las ejecuciones, localización y nombre del ejecutable binario, etc.

Los *scripts* son archivos de texto plano, y normalmente poseen extensión del tipo ".sh" (de ***sh****ell*).

Por otro lado hay que respetar la carga de trabajo del *front end*, nodo del *cluster* encargado de gestionar la interacción usuario-*cluster*. Para ello, en caso de necesitar ejecutarse algún tipo de programa en Calderón de manera interactiva (sin usar algún *script*, y tras haber recibido permiso de la administración del *cluster*) el usuario debe conectarse a un nodo interactivo con el comando necesario (ver más adelante). En el *front end* no se debe por tanto ejecutar directamente programas ni compilar código, ya que dicha carga puede retrasar y obstaculizar las funciones básicas y prioritarias de Calderón (gestión de la cola, acceso a otros usuarios, funcionamiento de su propio sistema operativo, etc). El *front end* debe ser usado para añadir trabajos (por medio de los ya mencionados *scripts*) a la cola de trabajos de Calderón, gestionar las cuentas de usuario, los archivos de trabajo, etc.

2.1.1.- Conexión con Calderón

Para poder conectarse al *cluster* se usará el protocolo ***S****ecure* ***Sh****ell* (SSH); es necesario introducir el siguiente código en la *shell*:

```
ssh -X nombre_de_usuario@calderon.atc.unican.es
```

Comando 2. Este comando en la *shell* de Unix, teniendo acceso a Internet, conectará al usuario al *cluster* Calderón, del grupo ATC de la Universidad de Cantabria.

A continuación, tras haber pedido acceso a Calderón, éste pedirá la contraseña de usuario.

Del mismo modo, se puede acceder al *cluster* por medio de una interfaz gráfica aportada por el sistema operativo. Ésta ha sido usada por el autor con la versión de Unix Ubuntu 9.10.

Imagen 4. Interfaz gráfica del Sistema Operativo Unix Ubuntu 9.10 para conectarse al *cluster* por medio del protocolo SSH. Éstos son los datos del autor.

La interfaz gráfica puede ser muy útil a la hora de gestionar archivos de manera visual con el uso de ventanas, pero no será útil a la hora de gestionar la cola de trabajo de Calderón, ya que se necesita usar la *shell* para ello.

2.1.2.- Conexión a un nodo de Calderón

En ocasiones, el usuario querrá conectarse a algún nodo en concreto de Calderón, tanto para compilar como para ejecutar un programa de manera interactiva. En este caso, una vez conectado al *cluster*, se han de introducir los siguientes comandos:

```
qrsh -l comp
```

Comando 3.a. Esta expresión permite acceder a un nodo de compilación del *cluster* una vez conectado a él. Dada la naturaleza del *cluster*, Calderón dispone de una muy amplia colección de compiladores.

Hay que tener en cuenta que es muy importante trabajar con ejecutables compilados en Calderón y no con ejecutables compilados en otras máquinas y copiadas después en el *cluster* (especialmente con otros sistemas operativos), ya que la arquitectura del computador es muy posible que difiera de la del *cluster*.

```
qrsh -l inter
```

Comando 3.b. Para conectarse a un nodo de ejecución interactiva de Calderón, el usuario, tras haberse conectado a él, deberá ejecutar esta orden. Una vez hecho esto, el gestor de colas le permitirá acceder a un nodo disponible para ser usado. En ningún caso estas ejecuciones pueden exceder los 35 minutos.

A cualquiera de estas dos expresiones se las puede añadir un sufijo para especificar en cuál de los múltiples nodos de compilación o nodos interactivos se desea establecer una conexión. Para más información, consultar el Anexo I.

Desde estos nodos de ejecución interactiva no se pueden gestionar los trabajos de la cola de Calderón (ni consultar la lista de la cola, ni añadir trabajos, ni cancelarlos, etc). Son nodos disponibles únicamente para poder ejecutar pequeñas aplicaciones.

En este proyecto se ejecutarán algunas aplicaciones creadas por el autor por medio de estos nodos, pero para poder ejecutar MAST en Calderón, se deberá hacer por medio del uso de los *scripts* apropiados en el *front end*.

2.1.3.- Desconexión de Calderón o de un nodo

Para poder desconectar la sesión del nodo actual y regresar al *front end*, o para desconectar la sesión actual con Calderón y regresar a la *shell* del computador cliente, se deberá ejecutar un sencillo comando:

```
exit
```

Comando 4. Ejecutando esta orden en la *shell*, estando el usuario conectado a un nodo del *cluster*, dirige al usuario de vuelta al *front end*, o en caso de ya encontrarse en él, cierra la sesión y devuelve al usuario a la *shell* de su computador.

2.1.4.- Gestión de la cola de trabajo de Calderón

Inevitablemente el usuario de Calderón deseará poner trabajos en la cola del *cluster* en algún momento. Para ello debe, por medio de los ya mencionados *scripts*, definir a Calderón la naturaleza de su trabajo. Los *scripts* son ejecutados en la *shell* como cualquier otro *script* o ejecutable en un computador normal:

```
./nombre_del_script.sh
```

Comando 5.a. El punto y barra indican al computador que "nombre_del_script.sh" debe ser ejecutado.

Por otro lado, este comando requiere que dentro del *script*, junto a las especificaciones del trabajo, se haya ubicado correctamente el comando que se explica a continuación. En caso de no haberlo incluido en el interior del *script*, éste se debe ejecutar de la siguiente forma:

```
qsub nombre_del_script.sh
```

Comando 5.b. El comando `qsub`, (del inglés ***q**ueue **sub**mit*, en español *entregar a la cola*) indica al gestor de colas de trabajo de Calderón que debe añadir a la cola el trabajo especificado en el *script* "nombre_de script.sh". Este comando debe ir incluido dentro del *script* si no es ejecutado de esta manera (sino por medio del comando 5.a).

En las aplicaciones creadas por el autor, los *scripts* de ejecución generados ya incluyen el comando

`qsub` en su interior, por lo que la manera de ejecutar los *scripts* será usando el comando 5.a.

Una vez añadido el trabajo a la cola, el gestor de la cola informará de que este trabajo (con un nombre asignado en el *script* de ejecución) ha sido añadido correctamente, e informará de su identificación en la cola (Calderón asigna un número de identificación a cada trabajo, este número será llamado *ID*).

Cuando el trabajo ya está en la cola, puede que el usuario esté interesado en conocer el estado de su trabajo dentro de la cola. Para poder ver la cola de trabajos de Calderón, ha de usarse la siguiente expresión:

```
qstat [-u nombre_usuario]
```

Comando 6. Por medio de este comando el usuario puede imprimir por pantalla información de la cola, desde la totalidad de trabajos que están siendo ejecutados en el *cluster*, sus usuarios y el tipo de trabajos, hasta el momento en que éstos fueron añadidos a la cola de Calderón. Si se ejecuta la orden (opcional) entre corchetes, se devolverá la información de la cola de dicho usuario.

```
job-ID  prior   name       user         state submit/start at     queue                          master ja-task-ID task-ID state cpu
tat failed
-----------------------------------------------------------------------------------------------------------------------------
----------
  63166 0.50500 inMast_1   jbrueda      r     12/02/2010 11:55:57 long_0-2_p1.q@compute-0-2      MASTER
  63167 0.50500 inMast_2   jbrueda      r     12/02/2010 11:55:57 long_0-3_p1.q@compute-0-3      MASTER
  63168 0.50500 inMast_3   jbrueda      r     12/02/2010 11:55:57 long_0-4_p1.q@compute-0-4      MASTER
  63169 0.50500 inMast_4   jbrueda      r     12/02/2010 11:55:57 long_0-12_p0.q@compute-0-12    MASTER
  63170 0.50500 inMast_5   jbrueda      r     12/02/2010 11:55:57 long_0-12_p1.q@compute-0-12    MASTER
  63171 0.50500 inMast_6   jbrueda      r     12/02/2010 11:55:58 long_0-12_p2.q@compute-0-12    MASTER
  63172 0.50500 inMast_7   jbrueda      r     12/02/2010 11:55:58 long_0-0_p0.q@compute-0-0      MASTER
  63173 0.50500 inMast_8   jbrueda      r     12/02/2010 11:55:59 long_0-7_p1.q@compute-0-7      MASTER
  63174 0.50500 inMast_9   jbrueda      r     12/02/2010 11:55:59 long_0-8_p1.q@compute-0-8      MASTER
  63175 0.50500 inMast_10  jbrueda      r     12/02/2010 11:56:00 long_0-9_p0.q@compute-0-9      MASTER
  63176 0.50500 inMast_11  jbrueda      r     12/02/2010 11:56:00 long_0-14_p0.q@compute-0-14    MASTER
  63177 0.50500 inMast_12  jbrueda      r     12/02/2010 11:56:00 long_0-15_p0.q@compute-0-15    MASTER
  63178 0.50500 inMast_13  jbrueda      r     12/02/2010 11:56:01 long_0-11_p1.q@compute-0-11    MASTER
  63179 0.50500 inMast_14  jbrueda      r     12/02/2010 11:56:01 long_0-14_p1.q@compute-0-14    MASTER
  63180 0.50500 inMast_15  jbrueda      r     12/02/2010 11:56:02 long_0-9_p3.q@compute-0-9      MASTER
  63181 0.50500 inMast_16  jbrueda      r     12/02/2010 11:56:02 long_0-11_p3.q@compute-0-11    MASTER
  63182 0.50500 inMast_17  jbrueda      r     12/02/2010 11:56:02 long_0-15_p3.q@compute-0-15    MASTER
  63183 0.50500 inMast_18  jbrueda      r     12/02/2010 11:56:03 long_0-13_p1.q@compute-0-13    MASTER
  63184 0.50500 inMast_19  jbrueda      r     12/02/2010 11:56:03 long_0-13_p3.q@compute-0-13    MASTER
  63185 0.50500 inMast_20  jbrueda      qw    12/02/2010 11:56:03
  63186 0.50500 inMast_21  jbrueda      qw    12/02/2010 11:56:04
  63187 0.50500 inMast_22  jbrueda      qw    12/02/2010 11:56:04
  63188 0.50500 inMast_23  jbrueda      qw    12/02/2010 11:56:04
  63189 0.50500 inMast_24  jbrueda      qw    12/02/2010 11:56:04
  63190 0.50500 inMast_25  jbrueda      qw    12/02/2010 11:56:05
  63191 0.50500 inMast_26  jbrueda      qw    12/02/2010 11:56:05
  63192 0.50500 inMast_27  jbrueda      qw    12/02/2010 11:56:05
  63193 0.50500 inMast_28  jbrueda      qw    12/02/2010 11:56:06
  63194 0.50500 inMast_29  jbrueda      qw    12/02/2010 11:56:06
```

Imagen 5. Captura de pantalla de la cola de trabajos de Calderón.

La lista de la imagen 5 sólo muestra los trabajos del usuario (jbrueda) después de usar el comando 6. Las columnas muestran, de izquierda a derecha: **1)** El número de identificación del trabajo en el gestor de colas (*ID*). En la lista los procesos están ordenados por este valor. Calderón asigna esta *ID* de manera progresiva a cada trabajo. **2)** Prioridad interna (valor numérico, 0.50500 en la imagen) que usa Calderón para gestionar su cola de trabajos (dependiendo de si son trabajos paralelizables, la duración de dichos trabajos, el tipo de usuario, etc). **3)** Nombre del proceso en el gestor de

colas, definido en el script de ejecución. **4)** Nombre del usuario cuyo trabajo está siendo gestionado. **5)** Estado del trabajo, "`r`" implica que está trabajando, "`qw`" implica que está esperando en la cola para ser ejecutado (ver tabla 1). **6)** Fecha y hora de entrega a la cola de dicho trabajo (cuando se ha ejecutado el comando `qsub` sobre dicho trabajo). **7)** Código del procesador y tipo de ejecución (en términos de tiempo de ejecución) del trabajo. **8)** Tipo de procesador, MASTER es un procesador autosuficiente, en caso de ser un procesador de tipo SLAVE implicaría que depende de un procesador MASTER y que se trata de un trabajo paralelizado (todos los trabajos de la imagen son secuenciales, por lo que los procesos se ejecutan en procesadores MASTER).

Junto al trabajo aparecerá un código que indicará el estado del trabajo, estos son los 3 tipos más comunes de dichos estados:

Código	Estado
`Eqw`	*Error queue waiting*, error esperando en la cola
`qw`	*Queue waiting*, esperando en la cola
`r`	*Running*, ejecutándose

Tabla 1. Códigos del estado de los trabajos en la cola de Calderón.

Normalmente los errores en la cola se deben a errores en el *script* de ejecución, un ejecutable corrupto o no ejecutable en Calderón (¿compilado en otro computador?) o porque ha sido mal añadido a la cola (errores en el uso de los comando 5.a ó 5.b).

El estado correspondiente a la espera en la cola ("`qw`") implica que el grupo de nodos al que ha sido añadido el trabajo, está totalmente ocupado, por lo que el gestor de colas retiene dicho trabajo a la espera de un procesador (o varios en caso de ser un trabajo paralelo) libre.

Por otro lado, a la hora de gestionar los trabajos, el usuario puede querer abortar una ejecución, bien porque ésta se encuentra en estado de error ("`Eqw`") y no será ejecutado (quedando en la cola de manera crónica e inerte), o porque la ejecución ya no es necesaria. Para poder eliminar un trabajo de la cola, se debe usar el siguiente comando:

```
qdel ID_del_trabajo
```

Comando 7. Comando que permite eliminar de la cola de trabajos de Calderón un trabajo con una *ID* conocida.

2.1.5.- Intercambio de archivos *cluster*-máquina cliente

En ocasiones se requerirá intercambiar algún archivo entre el *cluster* y el computador desde el que se encuentra el usuario. Lo más sencillo en estos casos puede ser usar el interfaz gráfico que se vio en 2.1.1, ya que crea una sesión de ventanas como si el mismo *cluster* fuese una ventana del mismo computador cliente.

Si no se puede iniciar una sesión de ventanas, o no se desea, se puede usar el protocolo ***S**ecure **C**o**p**y* (SCP) en la *shell*. El uso de este programa SCP se llevará a cabo con la siguiente sintaxis:

```
scp usuario@host:directorio/archivo_origen archivo_destino
```

```
scp archivo_origen usuario@host:directorio/archivo_destino
```

Comandos 8a y 8b. Comandos equivalentes para intercambiar archivos entre distintos equipos conectados mediante el protocolo Secure *Shell*. La sintaxis es muy similar a la del uso del comando `cp` en la *shell* de Unix, que sirve para copiar archivos dentro de la misma máquina.

Capítulo 3
Herramientas *software*

3.1.- Finalidad del *software*

Para entender la razón del *software* creado para este proyecto hay que analizar el procedimiento que se va a seguir cuando se ejecute una batería de análisis.

Existe la necesidad de estudiar a fondo cómo se comporta MAST cuando se ejecuta alguna de sus herramientas. Para estudiar esto, se ejecutan muchos sistemas de tiempo real de prueba. Éstos pueden tener una carga cual sea, desde el 0% para un sistema totalmente libre hasta un 100%, caso en el que el sistema está totalmente saturado. Esta carga, como ya se ha comentado, se denomina "utilización" y será descrita en el siguiente apartado. Mientras tanto basta con saber que se utilizará esta carga como variable controlada en rangos normalmente desde el 1% hasta el 99% (valores del 100% o mayores directamente no son analizados por MAST).

Por tanto, se precisa de un método para generar modelos MAST para sistemas de tiempo real con una utilización que sea uno. Este modelo MAST servirá como fichero semilla desde el que poder generar cualquier otro fichero con una utilización determinada.

Ahí surge la primera necesidad de crear un *software* que cumpla con ese proceso. Sistematizar la creación de modelos MAST, de tal manera que el usuario sólo tenga que controlar el menor número de variables de dicho sistema de tiempo real, tales como el número de procesadores o de transacciones. Una vez generado ese modelo MAST, reducir o ampliar su utilización al 1% con algún algoritmo. Y una vez conseguido ese fichero semilla, generar copias de dicho fichero cambiando los datos necesarios para que su utilización sean todas las posibles en el intervalo del 1% al 99%, pero manteniendo la arquitectura del modelo semilla.

Pero tras haber creado la serie de *inputs*, aparece una nueva necesidad, construir una herramienta que sea capaz de crear los *scripts* de ejecución de Calderón, y una vez programada, sistematizarla de manera que también pueda crear series enteras, es decir, por cada *input* de MAST, crear un *script* personalizado del trabajo.

Tras haber creado parejas de *inputs-scripts* el usuario se encuentra con cerca de 200 archivos que debe gestionar manualmente. Por ello también se añadió una herramienta creadora de otros *scripts*, distintos a los de ejecución de Calderón, para que con sólo ejecutarlos, las órdenes básicas se ejecuten de una sola vez para todos los archivos. Estos *scripts* se explican y analizan más adelante.

Por último, existen dos herramientas creadas para poder leer varios archivos rápidamente. Estos archivos son los archivos de salida de MAST y los archivos que informan al usuario del tiempo empleado por cada ejecución individual. De esta manera, la tediosa labor de abrir cada archivo de

salida y anotar la información importante que hay en él se simplifica de manera sustancial.

Todas estas herramientas serán descritas a continuación. Sin embargo hay que tener en cuenta que alguna de ellas no siempre será necesaria, ya que existe una herramienta que engloba varias de ellas y las sistematiza, mientras que las herramientas englobadas están diseñadas para crear *inputs* o *scripts* individuales (para casos y ejecuciones aisladas).

3.1.1.- Compilación del *software*

Para poder crear ejecutables a partir de ficheros de código, es necesario compilar dicho código. No es correcto usar ejecutables compilados en máquinas externas ya que Calderón posee una arquitectura especial, además de que al mover ejecutables de un sistema operativo a otro puede acarrear problemas de ejecución.

Para compilarlo el usuario deberá conectarse a un nodo de compilación del *cluster* (comando 3.a del apartado 2.1.2), y aquí ejecutar los comandos necesarios.

Para compilar tanto MAST como las herramientas diseñadas para este proyecto, se deben ejecutar dos *scripts*, cada uno en su paquete correspondiente, ambos con el nombre *compile.sh*.

El código usado para compilar este código Java y generar los ejecutables de extensión *.jar* se muestra en el Anexo II, así como el listado de archivos *.java* necesarios.

3.1.2.- Ejecución de *scripts*

Para poder ejecutar los *scripts* con los que se va a trabajar, es necesario que éstos posean permiso de ejecución, para ello, si no se posee tal permiso, el administrador, o alguien con privilegio para conceder dichos permisos, deberá ejecutar la siguiente orden:

```
chmod ugo+x nombre_del_script.sh
```

Comando 9. Con esta orden (`chmod`) se le concede (`+`) al *script* (`nombre_del_script.sh`) el permiso de ejecución (`x`) para cualquier tipo de usuario ("`u`" para el mismo usuario que le ha dado permisos, "`g`" para los usuarios del grupo de dicho usuario, y "`o`" para el resto de usuarios).

Una vez dotado del permiso de ejecución, se puede ejecutar usando la siguiente orden:

```
./nombre_del_script.sh
```

Comando 10. Comando usado para ejecutar el código del *script* `nombre_del_script.sh`.

3.1.3.- Ejecución de programas en Java

Una vez se disponga de un ejecutable Java, creado como se ha visto en el apartado 3.1.1, éste se puede ejecutar usando la siguiente orden:

```
java -jar Ejecutable.jar [parámetros]
```

Comando 11. Al usar esta orden se ejecuta el `Ejecutable.jar`, sin necesidad de que éste posea permiso de ejecución, ya que al ser un ejecutable, esto es algo intrínseco a él, a diferecia de

un *script*, que es un archivo de texto con órdenes usadas en la *shell* de Unix. Los parámetros son datos no obligatorios que solamente a veces pueden tener sentido en la ejecución del programa.

3.2.- Utilización, variable de MAST

La herramienta mast_analysis puede analizar distintos sistemas de tiempo real y calcular distintas variables de trabajo, la más importante para este proyecto, junto a la planificación, es la utilización.

La utilización de un procesador es la forma de medir la carga de trabajo de dicho procesador de manera porcentual. El modo de calcularlo requiere conocer los tiempos de peor caso, es decir, las duraciones más pesimistas, de las operaciones de dicho procesador y los periodos de las transacciones en que se encuentran dichas operaciones. También hay que tener en cuenta la velocidad v_i de los procesadores. De esta manera se puede expresar la utilización como:

$$Utilización_i = \sum_j \frac{t_{ij}}{T(t_{ij}) \times v_i} \times 100 \qquad \textbf{[1]}$$

Expresión [1]. La utilización del procesador *i*-ésimo se obtiene por medio de la duración de todas las operaciones (t_{ij}) y los periodos ($T(t_{ij})$) con los que se repiten dichas operaciones en las transacciones en que se encuentran. Este valor es porcentual, valores menores que 0 ó mayores que 100, no tienen por tanto sentido.

MAST considera a la red y sus operaciones de comunicación como un procesador en sí mismo. Por tanto se puede calcular también la utilización de este elemento. Pero en sistemas distribuidos, cuando se considere la utilización del sistema, se ignorará esta utilización y se calculará la media de utilizaciones de todos los procesadores (en sistemas mono-procesador, la utilización será, evidentemente, la del único procesador) y se tomará esta media como la utilización del sistema.

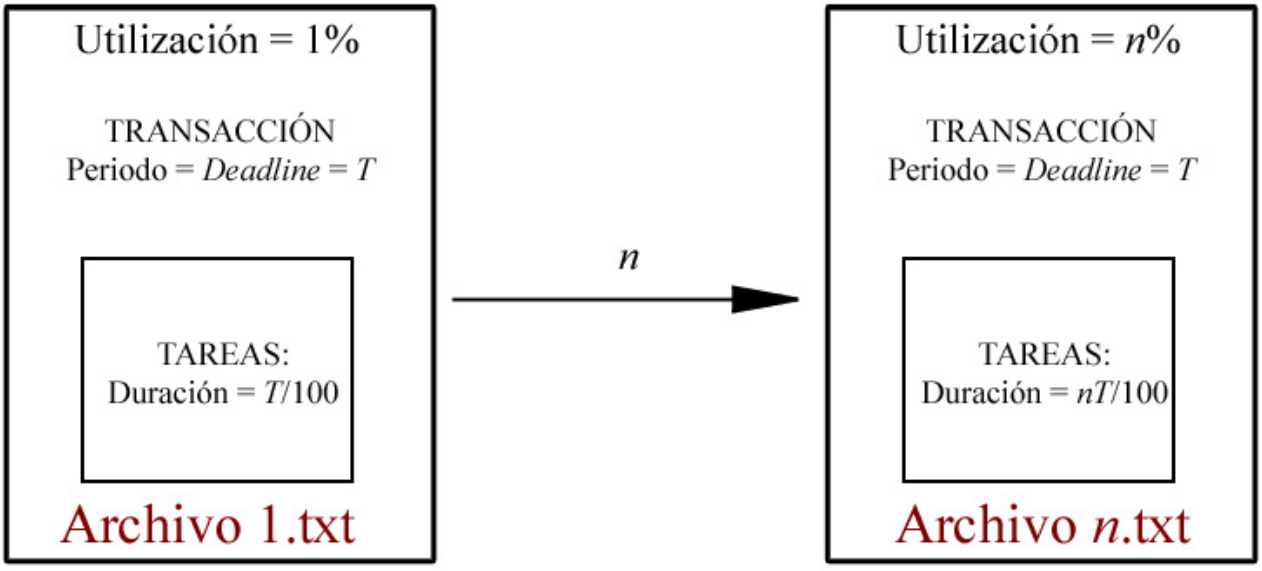


Figura 3. Esquema de cómo a partir de un modelo MAST semilla de una utilización del 1% se generan los modelos MAST *n*-ésimos de utilización *n*, creando una copia y reescalando las duraciones de todas las tareas.

3.3.- Extensiones de los ficheros

Debido a la gran cantidad de ficheros distintos con los que se va a trabajar, se explicará aquí cuáles son estos tipos de ficheros, diferenciándolos en la extensión de su nombre, es decir, el conjunto de caracteres que aparece al final del nombre, tras el último punto:

- TXT – Archivos de modelos de MAST, aunque es una extensión muy común para el texto plano;
- SH – *Scripts* de distinto tipo. Hay que entender que un *script* es un conjunto de comandos de *shell* almacenadas en un fichero que se ejecutan al invocar a dicho fichero, por lo que los *scripts* de ejecución en Calderón y los *scripts* comunes son realmente el mismo tipo de fichero, pero con distintos objetivos (uno es usar el gestor de colas y el otro utilizar el sistema operativo);
- OUT – Referido a archivos de salida (*output*). MAST genera estos archivos con información del análisis.
- TIME – Archivos con la información del tiempo empleado por una ejecución;
- O*X* – Donde *X* es el número (de varios dígitos) que identifica al trabajo en el gestor de colas de Calderón, el *ID*. Se puede observar en la imagen 5 Estos archivos son generados por Calderón con el texto que el programa ejecutado imprime normalmente en la pantalla (en la *shell*). Este fichero es útil para saber si el sistema ha sido planificable o no. La extensión suele ser modificada tras recopilar estos ficheros a algo más simple para facilitar su lectura (se verá más tarde en la descripción del *script* encargado de modificar estos ficheros) a una más sencilla (normalmente "*.O*");
- E*X* – Igual que en el caso anterior, sin embargo esta extensión se asocia a los archivos de error en vez de a los de salida. Si el trabajo se ha ejecutado satisfactoriamente el archivo debería estar vacío, en caso contrario se podrá encontrar una descripción del error en este documento;
- JAVA – Archivos de texto con el código JAVA;
- JAR – Extensión para los ejecutables creados en lenguaje JAVA.

3.4.- Creador de *inputs* para MAST (*InputMast.jar*)

La principal función de esta herramienta es crear de manera individual modelos MAST y trabajar con ellos. Esta herramienta presenta un menú al ser ejecutado en el que ofrece al usuario varias opciones en la creación, edición y cálculo de modelos MAST.

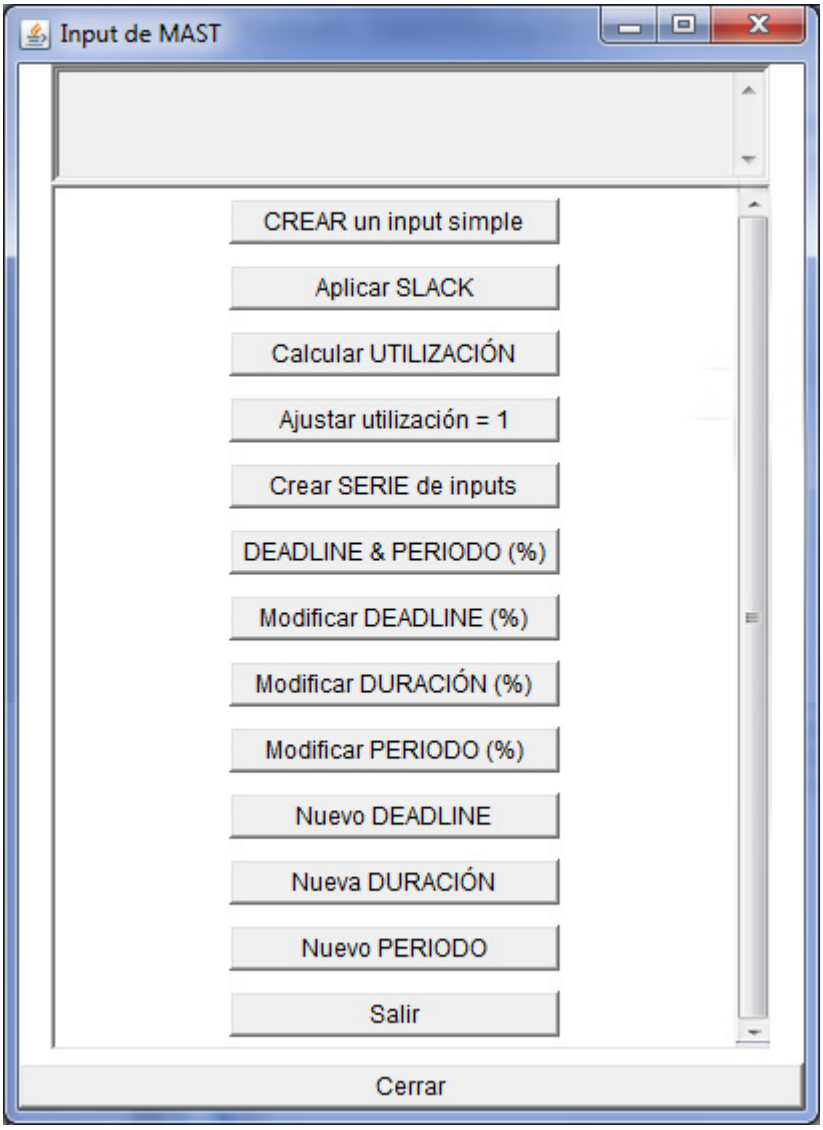


Imagen 6. Ventana del menú de la herramienta destinada a crear y editar *inputs* de MAST.

La primera y principal opción ofrece la posibilidad de crear un modelo MAST a partir de unos datos de entrada. Estos datos son:

1. Nombre del fichero (sin su extensión, que la presupone .txt);
2. Número de procesadores;
3. Número de transacciones (que debe ser mayor que el número de procesadores por cuestiones de arquitectura del programa);
4. Número de tareas que aporta cada procesador en cada transacción. Así, si se le indica que trabaje con 2 procesadores y 5 transacciones, con 3 tareas por procesador en cada transacción, cada transacción tendrá en total 6 tareas, existiendo en total en dicho sistema de tiempo real 30 tareas diferentes;
5. Valor mínimo y máximo posible de los tiempos de peor caso de las tareas (en un intervalo cerrado), ya que estos tiempos serán aleatorios en dicho rango.

El programa a partir de ahí genera un modelo MAST con valores aleatorios en el resto de variables, como en los tiempos de peor caso (en el intervalo acordado), el periodo (con valores aleatorios en

un intervalo proporcionalmente mucho mayor que el de las duraciones de las tareas), o el del *deadline*, que es el producto del periodo con el del número de procesadores (esto sirve para generar sistemas que se pueden ajustar a la realidad). Las prioridades están acotadas entre los valores 1 y 32767 (valor predefinido en *gmasteditor*), pero asignados en función del *deadline* (las tareas más prioritarias son aquellas que en las transacciones tienen *deadlines* más inminentes).

Se resumen las opciones que ofrece el menú:

- `CREAR un script simple` – Herramienta de creación de modelos MAST ya explicada;
- `Aplicar SLACK` – Al indicarle el *slack* del análisis, la herramienta ajusta los tiempos de peor caso de las operaciones para que el sistema sea planificable con el máximo tamaño de estos tiempos de peor caso posible;
- `Calcular UTILIZACIÓN` – Esta opción calcula la utilización del sistema que se le indique.
- `Ajustar utilización = 1` – La herramienta modifica un modelo MAST de utilización cualquiera para que su nueva utilización sea exactamente de uno. Esto se consigue igualando la velocidad de cada procesador definido en el sistema a su utilización, con lo que se consigue una utilización de uno. De esta manera se genera un modelo de MAST semilla;
- `Crear SERIE de inputs` – Tras indicarle a esta herramienta cuál es el fichero semilla, generará una serie de *inputs* desde 2 hasta 99, ya que el correspondiente al 1% es el mismo semilla;
- `DEADLINE & PERIODO (%)` - A la herramienta se le indica el *input* con que va a trabajar y el porcentaje que va a reducirse o ampliarse las duraciones del periodo el plazo (*deadline*) de cada transacción;
- `Modificar DEADLINE (%)` - Similar a `DEADLINE & PERIODO (%)`, pero sólo se aplica al plazo;
- `Modificar DURACIÓN (%)` - Se varía el porcentaje deseado cada tiempo de peor caso de cada operación del modelo MAST que se indique;
- `Modificar PERIODO (%)` - Idéntico a `Modificar DEADLINE (%)` pero aplicado sólo al periodo;
- `Nuevo DEADLINE` – Con esta herramienta se igualan todos los plazos (*deadlines*) de un sistema a un número indicado;
- `Nuevo DURACIÓN` – Igual que en `Nuevo DEADLINE`, pero aplicado a los tiempos de peor caso de las operaciones;
- `Nuevo PERIODO` – Igual que en `Nuevo DEADLINE`, pero aplicado a los periodos;
- `Salir` – Termina de ejecutar el programa iterativamente (también se puede pulsar la opción de `cerrar`).

Las opciones usadas para generar una serie de modelos de MAST desde el 1% al 99% son sólo 3 de esas opciones (“`CREAR un input simple`”, “`Ajustar utilización = 1`”, “`Crear SERIE de inputs`”), aunque para crear la serie completa se creó otra herramienta llamada *Principal.jar* que sistematiza este proceso y otros que aún no se han explicado.

3.5.- Creador de *scripts* de ejecución en Calderón (*ScriptCalderon.jar*)

Como se ha explicado, para poder poner en el gestor de colas de trabajos de Calderón un programa (y que éste sea después ejecutado), es necesario elaborar un *script* de ejecución con algunos datos que el gestor necesita para poder trabajar con dicho programa. Y esta herramienta es la apropiada para crear un *script* individual.

La edición de estos ficheros es a veces algo tedioso. Muchas veces se puede trabajar con los mismos *scripts* una y otra vez teniendo sólo que editar alguna línea que haga referencia a las herramientas u opciones usadas. Aunque esta herramienta está ideada para crear los *scripts* de manera individual, con la herramienta *Principal.jar* se podrán crear series enteras de *scripts*, lo que muchas veces haga más sencillo crear la serie de nuevo a editar todos y cada uno de los *scripts* ya existentes para adaptarlos al siguiente trabajo.

Esta herramienta permite crear *scripts* para ejecutar MAST; no sirve para ejecutar otro tipo de ejecutables, aunque los *scripts* son documentos de texto plano que pueden ser editados, y por tanto, cambiando muchas veces la última línea de código del *script* se puede adaptar para ejecutar otros tipos de ejecutables.

El programa pide a través de la *shell* (con eventuales menús en ventanas independientes) los datos necesarios para crear dicho *script*. Los datos que requiere son:

- Nombre del fichero;
- Directorio en que se encuentra el ejecutable;
- Directorio en que se desea guardar el archivo de salida (se recomienda usar el mismo que el ejecutable);
- Directorio en que se desea guardar el archivo de error (se recomienda usar el mismo que el ejecutable);
- El correo electrónico al que se informará del progreso del trabajo en la cola;
- En qué casos el usuario desea recibir información por correo electrónico del progreso de trabajo;
- Duración del trabajo y grupo de procesadores (los límites temporales son menos de 35 minutos para trabajos interactivos y menos de 24 horas para los trabajos cortos, la información relativa a los grupo aparece en el Anexo I) *;
- Nombre del trabajo en la cola, es importante que no empiece por un carácter numérico, ya que Calderón lo puede confundir con variables internas y puede dar lugar a errores;
- Nombre del modelo de entrada de MAST (*input*);
- Nombre del documento de salida de MAST (*output*);
- Nombre del documento de salida del tiempo del trabajo;
- Lista con los directorios de las librerías necesarias para el ejecutable (MAST sólo necesita una, que ya aparece de manera predefinida en el programa: "`usr/lib`");
- Herramienta para ejecutar el trabajo;
- Opciones, si hiciesen falta, para la ejecución del trabajo;

* Si el trabajo sobrepasa los límites temporales definidos en el *script*, éste será abortado por el gestor de colas.

Una vez terminado, el programa genera un fichero de texto cuyo nombre es “nombre del fichero.sh”.

A continuación se muestra un ejemplo de *script* para una ejecución típica de MAST:

```
#!/bin/sh
# ENVIAMOS el TRABAJO AL SISTEMA DE COLAS
shell="/bin/bash"
path="workspace/ejemplo/"
output_path="workspace/ejemplo/salida/"
error_path="workspace/ejemplo/error/"
mail="javier.barba@alumnos.unican.es"
complex="short-g2"
job_name="ejecutable-ejemplo"
input="input.txt"
output="output.out"
#-------------------------------------------------
qsub << eor
#$ -S $shell
#$ -N $job_name
#$ -o /dev/null
#$ -e /dev/null
#$ -M $mail
#$ -m be
#$ -l $complex
# PERMISOS SGEEE-AFS
kinit -k -t /home/Keytab/krb5.keytab.$USER $USER
aklog -c ATC.UNICAN.ES -k ATC.UNICAN.ES
# VARIABLES DE ENTORNO PARA EL TRABAJO
export LD_LIBRARY_PATH="usr/lib"
# EJECUTABLE trabajo SIMPLE
#-------------------------------------------------------
# Encapsulado del trabajo
#
if [ true ]; then
      #-----------------------------------------------------------
      # pon aquí el path absoluto de ejecutable con sus parametros: p.e
      cd $path
      (time workspace/ejemplo/mast_analysis holistic -s $input
      $output)2>../time/tiempo-ejemplo.time
      #-----------------------------------------------------------
fi 1>\$output_path\$REQUEST.o\$JOB_ID 2>\$error_path\$REQUEST.e\$JOB_ID
#############################################################################
###############
eor
```

Las líneas que comienzan por una almohadilla (#) son comentarios para facilitar la comprensión del *script*, excepto algunas líneas en que a la almohadilla la suceden otros caracteres especiales, como el de dólar o la exclamación cerrada en el ejemplo. Las palabras que empiezan por el carácter “`$`” son variables del *script*.

Las primeras líneas del ejemplo sirven para declarar las variables, empezando por el tipo de *shell* de Unix (en este caso *bash*) con que se ha escrito el *script* hasta el nombre del *output*. A continuación se usa el comando `qsub` para enviar al gestor de colas de trabajos de Calderón la información de la ejecución que se encuentra a continuación (desde “-S” para el tipo de *shell* hasta “-l”

para definir el tipo de complejidad del proceso, en cuanto a tiempo y grupo de procesadores). Tras ello, se escriben unas líneas que aportan información adicional a la máquina para ejecutar el programa, como son las librerías necesarias para la ejecución o información de los permisos de trabajo. Por último se ejecutan las líneas que se encuentran encapsuladas dentro del comando "*if* [*true*]", que son las necesarias para ejecutar mast_analysis de manera cronometrada (time), usando la herramienta *holistic*, con la opción de *slack* (-s) a partir del *input* indicado. El archivo *tiempo-ejemplo.time* del directorio *time/* almacena la información del tiempo cronometrado de la ejecución.

Editando correctamente este ejemplo se puede usar para otros usos de MAST e incluso para otras ejecuciones secuenciales de otros programas.

3.6.- Lector de *outputs* (***BuscaSchedulable.jar***)

Una vez terminada la ejecución de todos los procesos, Calderón generará varios archivos de salida mencionados en el apartado 3.3, cuya extensión es *.oX*, siendo *X* el código *ID* del trabajo *i*-ésimo.

Para poder ejecutar esta herramienta, todos estos archivos deberán tener la misma extensión .o, por lo que se deben renombrar todos esos ficheros eliminando el *ID* del nombre. Para ello se puede usar otra herramienta descrita más tarde, llamada *limpiador.sh*.

En estos archivos se encuentra la información que el usuario vería impresa en la *shell* si ejecutase mast_analysis en su propio computador. Entre estas líneas impresas en la pantalla se encuentra una línea esencial para el estudio de estos análisis, la que indica si el trabajo no es planificable ("`Final analysis status: NOT-SCHEDULABLE`") o sí lo es ("`The system is schedulable`").

Revisar cada documento de salida en busca de la línea que informe de este dato es bastante tedioso. Por ello se programó esta aplicación que busca en un determinado directorio entre todos los archivos de salida la línea que identifica cada trabajo como sí o no planificable.

Al ejecutar la herramienta se pide la ubicación del directorio relativo a la ubicación actual de este ejecutable, se dejará en blanco si el ejecutable se encuentra en el mismo directorio en que se encuentran los archivos. También se pide el prefijo que tienen los archivos, es decir, toda la serie comenzará con el mismo prefijo y terminará con el número de la serie que le corresponde (prefijo*i*.o). Este prefijo será el que se ingrese al generar los *scripts* de ejecución de Calderón (*nombre del trabajo en la cola*).

La herramienta informará por pantalla de todas las soluciones de cada elemento de la serie.

Ésta hará un barrido desde el 1% hasta el 99% en saltos de la unidad buscando dichos archivos. Si en vez de trabajar con series de uno en uno, se ha trabajado con series de distinto intervalo, funcionará correctamente también, pero los elementos de la serie que no se localicen aparecerán como "*no encontrados*".

Véase un ejemplo de lo que puede encontrarse el usuario al ejecutar esta aplicación con unos

datos de salida ubicados en la carpeta "*Salidas*" del directorio actual, y con un prefijo "*BSE_*":

```
Nombre del directorio (en blanco para usar el directorio actual):       Salidas/
Prefijo de los outputs:         BSE_
1) PLANIFICABLE
2) PLANIFICABLE
      ...
73) PLANIFICABLE
74) PLANIFICABLE
75) NO PLANIFICABLE
76) PLANIFICABLE
77) PLANIFICABLE
78) PLANIFICABLE
79) NO PLANIFICABLE
80) NO PLANIFICABLE
81) NO PLANIFICABLE
82) NO PLANIFICABLE
83) PLANIFICABLE
84) NO PLANIFICABLE
      ...
95) NO PLANIFICABLE
96) NO PLANIFICABLE
97) no encontrado
98) NO PLANIFICABLE
99) NO PLANIFICABLE
```

Imagen 7. Captura de pantalla de la ejecución de la herramienta buscadora de trabajos planificables. Cada modelo MAST ejecutado está representado en cada línea, correspondientemente el índice con la utilización de dicho sistema. Las 3 posibles soluciones para cada sistema son: "`PLANIFICABLE`", "`NO PLANIFICABLE`" o "`no encontrado`", lo que quiere decir que en el documento no se hace mención a la planificación del trabajo (es un documento incorrecto, corrupto o que el trabajo no haya terminado aún) o porque directamente no existe tal fichero. Los puntos suspensivos de la imagen no aparecen en la imagen original, aquí sustituyen a todos los valores intermedios para simplificar la imagen.

3.7.- Lector de tiempos (*LectorTiempos.jar*)

Una vez finalizada la ejecución de la serie de trabajos, se habrán generado no solamente los archivos de salida de MAST (en todos los directorios de trabajo), sino también los que genera Calderón (usados con la herramienta anteriormente explicada, *BuscaSchedulable.jar*) y los archivos de salida de los tiempos (con extensión *.time*).

Estos archivos de salida de tiempos tienen una estructura en la que se almacenen 3 tiempos distintos, cada uno con una cabecera diferente en cada línea:

Cabecera	Nombre	Definición
`real`	Tiempo real de la CPU	Verdadera cantidad de tiempo que tarda el proceso en ser ejecutado. Tiene en cuenta no sólo el tiempo que tarda en operar, sino también por otras acciones, como el tiempo que es bloqueado por el sistema.
`user`	Tiempo de usuario	Tiempo que toma el trabajo en "modo usuario", es decir, sin interactuar con el sistema o con bloqueos externos. Este tiempo es el tiempo intrínseco de ejecución que se tendrá en cuenta.
`sys`	Tiempo del sistema	Tiempo que el sistema usa para manejar el trabajo, "modo *kernel*".

Tabla 2. Distintos tipos de tiempos medidos con el programa *time* de Unix. En este caso se tendrá en cuenta sólo el tiempo de usuario, ya que es el que muestra realmente el tiempo de ejecución del proceso intrínsecamente.

De manera similar al funcionamiento del programa *BuscaSchedulable.jar*, este programa buscará en un directorio indicado por el usuario, todos los archivos *n*-ésimos (desde 1 hasta 99 de manera predefinida) con la extensión *.time*. Si al ejecutar este programa se le introducen como parámetros dos números enteros, el programa entenderá que se trata de los valores mínimo y máximo del rango de utilizaciones y sólo rastreará e imprimirá en pantalla los valores comprendidos entre dichas cotas.

El programa leerá cada archivo y buscará la línea que corresponde al tiempo de usuario, recogerá el tiempo, que aparece en minutos más segundos (el tiempo tendrá esta forma: "`5m57.018s`", donde "`5m`" hace referencia a 5 minutos y "`57.018s`" serán 57 segundos con 18 milésimas, el programa lo convertirá a 357.018 segundos). A continuación listará en la *shell* de Unix en que haya sido ejecutado, todos los tiempos en segundos de cada elemento de la serie. Si este elemento no existiese, porque la serie no tomó intervalos de la unidad, sino mayores, el programa imprimirá este elemento sin ningún valor numérico en su tiempo.

3.8.- Programa Principal, creador de series de *inputs* y *scripts* (*Principal.jar*)

Para poder abordar grandes cantidades de ejecuciones, es necesario organizar correctamente el área de trabajo. Por ello, es necesario entender cómo se organizarán los archivos generados por esta herramienta, ya que al ejecutar tantos modelos MAST distintos es necesario crear varias copias de MAST, en distintas carpetas, con sus modelos MAST y sus *scripts* de ejecución.

Se llamará directorio de trabajo a la carpeta raíz en que se encuentren las herramientas. A partir de este directorio raíz se crearán distintos subdirectorios, la mayoría con un nombre numérico, identificando con dicho número (entero) la utilización del sistema que albergará. Otras carpetas almacenarán los archivos de tiempo de ejecución, las copias de seguridad de los modelos semilla, etc.

El proceso que seguirá esta herramienta será como indica la figura:

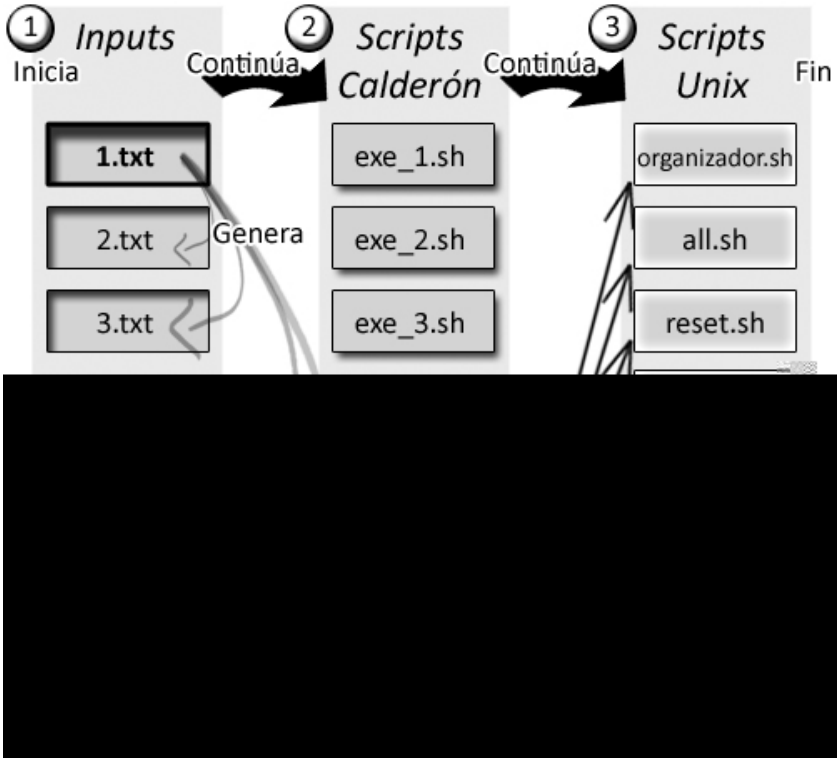


Imagen 8. Esquema del funcionamiento del programa Principal. De izquierda a derecha, comienzan creándose los modelos MAST según los datos que dé el usuario, se continúa con la creación de la serie de *scripts* de ejecución en Calderón y se terminan creando varios *scripts* en lenguaje *bash* para gestionar y facilitar el trabajo con todos los ficheros. Se explica a continuación en detalle.

1. Se inicia la ejecución creando un modelos MAST genérico con los mismos datos necesarios que para crear un *input* con la herramienta *InputMast.jar* (apartado 3.4);
2. Modificación del modelo MAST creado para que tenga utilización uno y convertirlo así en un modelo MAST semilla;
3. Se diseña la serie numérica de utilizaciones que se va a usar (número de inicio, de fin e intervalo de la serie);
4. Se guardan los valores de la serie en un archivo llamado *registro.txt*;
5. Creación de toda la serie de modelos con distintas utilizaciones enteras según el *registro.txt* a partir del archivo semilla;
6. Creación de los *scripts* de ejecución de cada trabajo (según los valores del *registro.txt*), muchos de los datos necesarios (ver apartado 3.5) son rellenados con información relativa al caso concreto que van a ejecutar, como el fichero de entrada, el directorio de trabajo, el nombre del trabajo en la cola (sólo es necesario indicarle el prefijo), etc. Se les nombrará "exe_*i*.sh".
7. Creación de un *script* (*organizacion.sh*) encargado de ordenar todos los archivos (consultando el *registro.txt*) una vez se termine la ejecución de este programa;
8. Creación de un *script* (*all.sh*)que ejecute todos los trabajos (consultando el *registro.txt*) de manera secuencial.
9. Creación de un *script* (*reset.sh*) que elimine los archivos de salida (de los trabajos indicados en *registro.txt*) y temporales, para evitar problemas de archivos mezclados;
10. Creación de un *script* (*backup.sh*) destinado a guardar copias del modelos MAST semilla en

un directorio cuyo nombre haga referencia a la fecha y hora en que se ha ejecutado;
11. Creación de un *script* (*limpiador.sh*) que renombre los ficheros de salida de tipo .O*X* para eliminar la parte *X* y poder usar la herramienta *LectorTiempos.jar*;
12. Fin de la ejecución.

Este proceso se lleva a cabo usando algunas de las herramientas explicadas anteriormente (*inputMast.jar* y *ScriptCalderon.jar*) de manera interna en este programa, además de nuevas partes destinadas a crear los *scripts* de ayuda y de gestión.

El esquema de la imagen 8 ayuda a entender mejor este proceso.

Una vez generados todos los modelos MAST, *scripts* de Calderón y los otros *scripts*, se trasladarán junto a una copia de mast_analysis al directorio de trabajo, si no se ha hecho ya o no se han generado directamente en este espacio.

Una vez con todos los archivos en el directorio de trabajo, se ejecutará el *script organizador.sh*, que ordenará todos los archivos en carpetas para poder ejecutar de manera ordenada todos los trabajos y poder almacenar los datos de salida.

Hay que recordar que para poder ejecutar los *scripts*, es necesario que éstos posean permiso de ejecución, como se vio en el apartado 3.1.2.

Los archivos, una vez ordenados, deberían estar organizados según se indica en la imagen 9.

Tras ordenar todos los ficheros, se pueden ejecutar usando el *script all.sh* desde el directorio de trabajo.

Una vez todos los trabajos han finalizado, se recopilan los distintos ficheros de salida que sean relevantes. En la carpeta de nombre *time/* se encontrarán los ficheros con la información de los tiempos de ejecución. También en la carpeta *time/* se encuentra otra carpeta de nombre *backup/* en la que se encontrarán otros subdirectorios con un archivo correspondiente al sistema semilla. En el directorio raíz del usuario se habrán generado los archivos de salida por pantalla de extensión .O*X* y .E*X*, que puede ser usados para averiguar cuáles de los sistemas ha sido planificable. En cada directorio de cada modelo MAST se encontrará el archivo de salida generado por MAST con información de cada sistema.

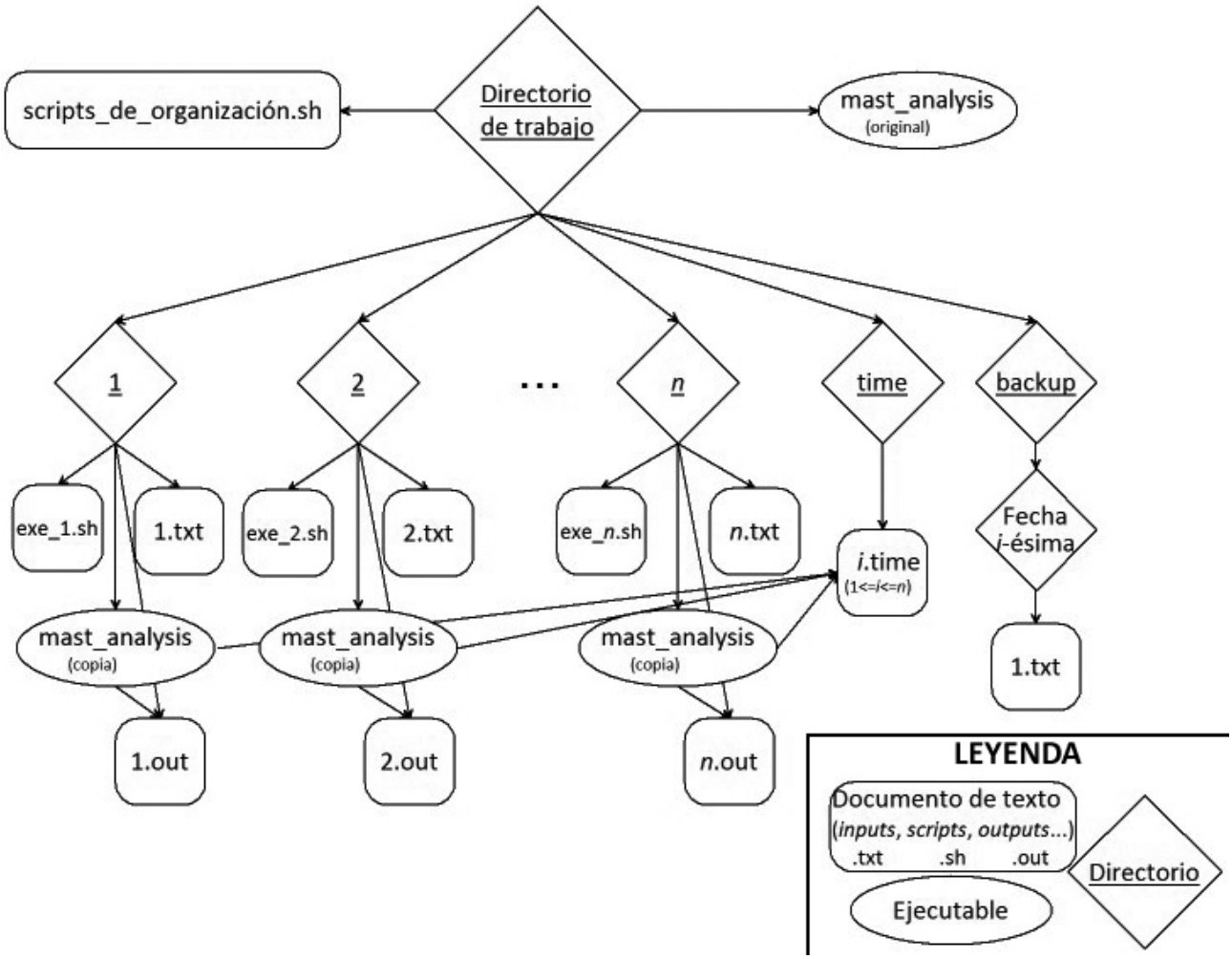


Figura 9. Esquema de la organización de directorios y ficheros en Calderón para la ejecución de MAST. Existe una carpeta para cada sistema de tiempo real de una determinada utilización (de ahí el nombre de cada directorio). Cada carpeta de trabajo posee una copia de mast_analysis, además de un *input* modelo de MAST y de un *script* de ejecución de Calderón. Existen dos subdirectorios distintos, *time/* y backup/, donde se almacenan los archivos del tiempo de ejecución y el fichero semilla, correspondientemente.

A continuación se listarán y analizarán los *scripts* que el programa *Principal.jar* crea al ser ejecutado, además de los necesarios para poner los ejecutables en la cola de trabajos de Calderón.

3.8.1.- *Script* de organización

A este *script* se le ha llamado *organizador.sh*.

Cuando se ejecute la herramienta *Principal.jar*, que enlaza la mayoría de estas herramientas, dicha herramienta generará una serie de *inputs* y *scripts* de ejecución asociados. Para ordenar esta gran cantidad de archivos, se puede ejecutar este *script*.

Las órdenes que ejecuta son las siguientes:

1. Elimina la carpeta *time/* y todo lo que contenga, ya que el análisis es nuevo y se sobres-

cribirá su contenido.
2. Se crea una nueva carpeta llamada *time/* y otra llamada *backup/*.
3. Se eliminan archivos temporales creados por la herramienta *Principal.jar.*
4. Para el archivo *i*-ésimo, elimina el directorio *i*-ésimo (de ejecuciones previas).
5. Se crea de nuevo la carpeta *í*-ésima vacía.
6. Se mueve el *input* y el *script* de ejecución *i*-ésimo al directorio *i*-ésimo.
7. Se da permiso de ejecución al *script* de ejecución *i*-ésimo.
8. Se hace una copia de mast_analysis del directorio raíz en el directorio *i*-ésimo.
9. Se dan permisos de ejecución a los *scripts* que ponen a todos los trabajos en el gestor de colas (explicado en el siguiente apartado) y el que crea una copia de seguridad de este modelo MAST semilla (explicado e el apartado 3.8.4).

Esta ejecución puede tardar algunos minutos, ya que debe copiar en torno a una centena de veces el ejecutable mast_analysis, que posee un tamaño de casi 20 megabytes.

Tras haber ejecutado este *script*, se tendrán todos los archivos ordenados y listos para ser puestos en el gestor de colas usando el siguiente *script*.

3.8.2.- *Script* para poner todos los trabajos en el gestor de colas de Calderón

Este *script* tiene por nombre *all.sh*.

Teniendo todos los *inputs* ordenados en subdirectorios cuyo nombre es el número entero de su utilización, junto al ejecutable mast_analysis y al *script* de ejecución (como se indica en el punto anterior y en el esquema de la figura 9), este *script* está listo para poder enviar al gestor de colas todos los trabajos. Las órdenes de dicho *script* son:

1. Eliminar los archivos de salida previos (incluyendo los de información de los tiempos *.time*) a esta ejecución para no confundir los nuevos archivos con los viejos. Para ello usa el *script* que se detalla en el siguiente punto (*reset.sh* 3.8.3).
2. Hace una copia del modelo MAST semilla (el correspondiente al 1%, de la carpeta llamada "1") utilizando el *script* definido más adelante (*backup.sh* 3.8.4).
3. Cambiar el directorio actual por el directorio *i*-ésimo.
4. Se ejecuta el *script* de ejecución del directorio actual.

A medida que los trabajos se van poniendo en la cola, el gestor informará de que el archivo *i*-ésimo ha sido enviado al gestor de la cola de trabajos de Calderón e informará de su *ID* en la cola. Este proceso suele tardar un menos de un minuto.

3.8.3.- *Script* de *reset*

Este *script* llamado *reset.sh* se encarga de eliminar los archivos de salida de ejecuciones previas para no equivocar estos *outputs* con los archivos de salida del análisis actual.

Las órdenes que sigue el *script* son:
1. Eliminar cada archivo de salida *.*out* del directorio *i*-ésimo del directorio de trabajo.
2. Eliminar todos los archivos de salida *.time* de la carpeta *time*.

3. Eliminar los archivos de salida y errores estándar de Calderón (correspondientes en el apartado 3.3 a los de extensión .O*X* y .E*X*).

La eliminación de estos archivos no debe hacerse a la ligera y se recomienda no ejecutar manualmente dicho *script* (sólo al ejecutar el *script all.sh*, ya que está implícito en la ejecución de dicho *script)*.

3.8.4.- *Script* de *backup*

El *script* se llama *backup.sh*.

La función de este *script* es la de crear en la carpeta *backup/* un subdirectorio cuyo nombre sea: `año-mes-día hora.minuto` (por ejemplo, para una ejecución en el último minuto del año 2010 sería `10-12-31 23.59`). Tras crear dicho directorio, se copiará en su interior el modelo MAST semilla *1.txt*.

De esta manera las carpetas estarán ordenadas temporalmente, y se podrá identificar más adelante la copia por la fecha en que fue ejecutada, y con el archivo matriz se puede generar toda la batería de *inputs*, creando una nueva serie de utilizaciones, usando la herramienta *InputMast.jar*.

3.8.5.- *Script* para cambiar la extensión de los archivos de salida por pantalla

Este último *script* se llama *limpiador.sh*, en referencia a la limpieza que hace en la extensión del nombre de los *outputs* que genera Calderón.

Como se ha visto, los archivos de salida pueden ser de dos tipos, los propios de MAST (.OUT) y los que genera Calderón (.O*X* y .E*X*) a partir del texto que genera el *software* por pantalla mientras es ejecutado. Este último texto se guarda en unos *outputs* de formato .O*X*, donde las *X* hacen referencia a un código numérico (*ID*), que si bien en bastantes ocasiones puede ser útil para diferenciar archivos entre ellos, es algo incómodo a la hora de sistematizar la lectura de la totalidad de estos *outputs* con herramientas *software* como la usada, *BuscaSchedulables.jar*.

Al ejecutar el *script* éste pedirá al usuario que escriba la ruta relativa a este *script* de la carpeta que almacena los archivos. Debe dejarse en blanco en caso de encontrarse en la misma carpeta, igual que al ejecutar *BuscaSchedulables.jar* o *LectorTimes.jar*.

Lo que hace el *script* es buscar todos los archivos en dicho directorio en cuyo nombre exista la cadena de caracteres "*.o*". Una vez hecho esto, renombrará cada uno de estos archivos cortando su nombre hasta dicho punto, por ejemplo, "Mast_*i.oX*" tendrá como nuevo nombre "Mast_*i.o*", mucho más cómodo para poder leerlo de manera sistematizada.

El uso de este *script* debería hacerse con cuidado, ya que el *script* no diferenciará los *outputs* de cualquier archivo ajeno a la ejecución cuyo nombre posea la cadena "*.o*", pudiendo de esta manera modificar archivos cuyo nombre no se desea alterar. Además se deberá evitar nombrar a los elementos de la serie en Calderón con puntos en el nombre, y menos con puntos seguidos de la vocal "*o*".

Capítulo 4
Ejecuciones de MAST en Calderón

4.1.- Paralelización de análisis de sistemas de tiempo real

De acuerdo a los objetivos enumerados en el apartado 1.3, la principal prioridad es disponer una herramienta para realizar grandes cantidades de análisis de las herramientas de MAST.

Se ha expuesto como ejemplo los 110000 análisis de ejemplo que se llevaron a cabo en la referencia **[6]** para poder verificar la validez de un nuevo algoritmo. En este caso se presenta un nuevo algoritmo heurístico llamado *HOSDA* que asigna nuevos plazos de planificación en un sistema de tiempo real si éste no es planificable. Éste método es una adaptación de *HOPA* (visto en el apartado 1.1.4) para sistemas con planificación EDF.

Aquí se analizarán y reasignarán las prioridades usando el método *HOPA* a algunos de los modelos MAST usados en dicho trabajo de la bibliografía **[6]** ligeramente variados, ya que hay que tener en cuenta que la arquitectura usada en estos modelos MAST es algo diferente a la que genera el *software* de este proyecto: sus transacciones están formadas por un número aleatorio de tareas, entre uno y el número de procesadores, lo que también implica que los *deadlines* de las transacciones sean, dependiendo de la cantidad de procesadores que intervienen en una transacción, un determinado múltiplo del periodo; además de que todas las prioridades son inicialmente uno, por lo que el sistema sólo puede ser planificable si se utiliza algún método de asignación de prioridades.

Se han usado algunos de los modelos MAST de ejemplos de dicha investigación para realizar los mismos análisis que realizaron sus autores en su momento. A través de la herramienta *InputMast.jar* se creó un modelo semilla y toda la serie de utilizaciones (reduciendo primero su utilización a uno y creando después la serie a partir de dicho archivo), que mantiene la arquitectura del sistema. Además se asignó a cada modelo una prioridad cualquiera y distinta a cada tarea, con el fin de facilitar los cálculos iniciales a MAST, con lo que se optimiza y reduce el tiempo de análisis de esta herramienta.

Los modelos MAST usados son de 3 tipos: Grande o BSE (*Big Size Example*), intermedio o ISE (*Intermediate Size Example*) y pequeño o SSE (*Small Size Example*), con las siguientes arquitecturas:

	SSE	ISE	BSE
Número de procesadores	3	5	8
Número de transacciones	6	8	12

Tabla 3. Arquitectura de los modelos MAST usados en esta parte del proyecto. Las transacciones están formadas por un número aleatorio de tareas entre 1 y el número de procesadores, por lo

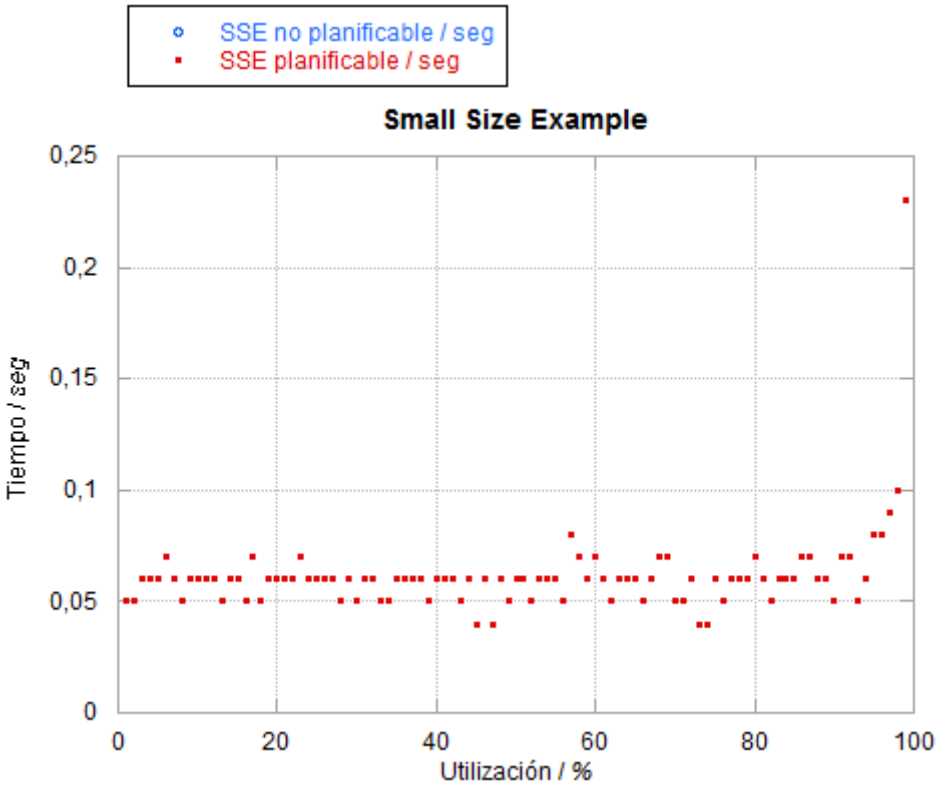
que como mucho, cada transacción contribuye en cada procesador con una sola tarea. Se van a analizar estos 3 ejemplos y sus correspondientes series una sola vez para comprobar los tiempos invertidos en los análisis, así como los valores máximos de planificación obtenidos.

Todos los análisis se llevaron a cabo con la herramienta *offset based optimized* y *HOPA* para asignar las prioridades. Los nuevos parámetros obtenidos al asignar nuevas prioridades fueron guardados junto a los ficheros de salida. El grupo de procesadores de Calderón utilizado fue el grupo 2.

4.1.1.- Ejemplo pequeño (*SSE*)

Este ejemplo, como el resto de ejemplos, fueron cedidos por los autores del artículo mencionado. Posee 3 procesadores y una utilización media del 10.27% (0.07%, 10.40% y 20.33% para cada uno de sus procesadores), por lo que en primer lugar se creó el modelo semilla cambiando la velocidad de cada procesador en función de su utilización para obtener una utilización de la unidad en cada procesador, dejando una utilización media del sistema de 1.

Una vez creado dicho modelo y toda la serie, se analizaron en Calderón los 99 modelos (el 17 de febrero del 2011 a las 16:52) y se obtuvieron las siguientes respuestas temporales:



Gráfica 1. Tiempos de cada análisis. Todos ellos concluyeron de manera planificable y sin incidencias especiales en los tiempos.

El tiempo medio para la serie es de 0.06 segundos (con un error de lectura de 0.01 segundos y una desviación estándar de 0.02).

Al ser el ejemplo más pequeño y simple de los 3, como ya se verá, es evidentemente el que menos

tiempo necesita para ser analizado. El tiempo total necesario para el análisis de los 99 sistemas fue de 6.02 segundos, y el máximo de todos ellos es de 0.23 segundos.

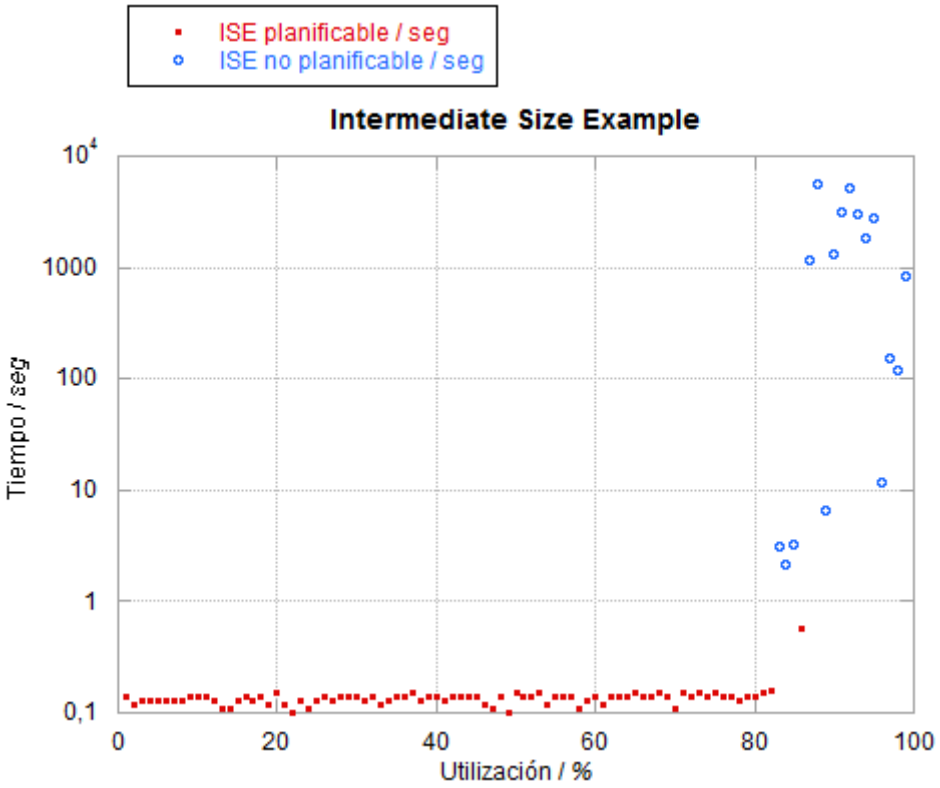
Aunque puede parecer que la ventaja del uso de un *cluster* en este caso no es relevante, hay que destacar que los tiempos obtenidos son para 99 sistemas, mientras que los test habituales pueden estar tres órdenes de magnitud por encima.

4.1.2.- Ejemplo intermedio (*ISE*)

En este caso el modelo original posee 5 procesadores con una utilización media del 10.46% (para los procesadores individualmente del 20.55%, 0.27%, 0.59%, 10.34% y 20.54%, que es bastante heterogéneo).

De nuevo se simplificó el sistema hasta reducirlo como ya se ha explicado a un sistema de utilización 1 y a una serie de modelos que son derivados de dicho sistema semilla.

En el análisis *offset based optimized* y asignación de prioridades *HOPA* en el grupo 2 de procesadores de Calderón (ejecutado el 17 de febrero del 2011 a las 16:53) se extraen los siguientes resultados:



Gráfica 2. Valores de los tiempos empleados en el análisis de cada modelo, expresado en escala logarítmica para apreciar mejor las diferencias pese a las diferentes magnitudes de los tiempos. En este caso aparecen por primera vez casos no planificables.

Además de poder ver la existencia de casos no planificables, es curioso encontrar sistemas que según el análisis sí son planificables cuando son casos de mayor utilización que otros no

planificables. Esta situación es bastante común cuando nos encontramos con cargas cercanas al límite de planificabilidad. El algoritmo heurístico HOPA podría encontrar soluciones mediante la variación de las constantes de las que depende, pero esta búsqueda está fuera de los objetivos de este proyecto.

Cabe destacar además la diferencia de tiempos necesaria según el sistema sea planificable o no planificable, ya que el algoritmo *HOPA* reitera el análisis hasta un máximo de 180 veces reasignando nuevas prioridades (a no ser que en uno de los análisis el sistema sea planificable, momento en el que el algoritmo finaliza).

	Suma de tiempos / *seg*	Media del tiempo / *seg*	Desviación estándar / *seg*	Tiempo máximo / *seg*
Sistemas planificables (83)	11.58	0.14	0.05	0.57
Sistemas no planificables (16)	25287.29	1600	1900	5538.61
Todos los sistemas (99)	25298.86	300	900	5538.61

Tabla 4. Tiempos de ejecución del conjunto de los sistemas de tiempo real del caso intermedio (*ISE*), valores medios con sus desviaciones estándar y tiempos máximos.

Como se puede observar, el tiempo necesario para analizar los 16 sistemas no planificables es de un total de 7 horas, nada que ver con los menos de 12 segundos para los otros 83 sistemas que sí son planificables.

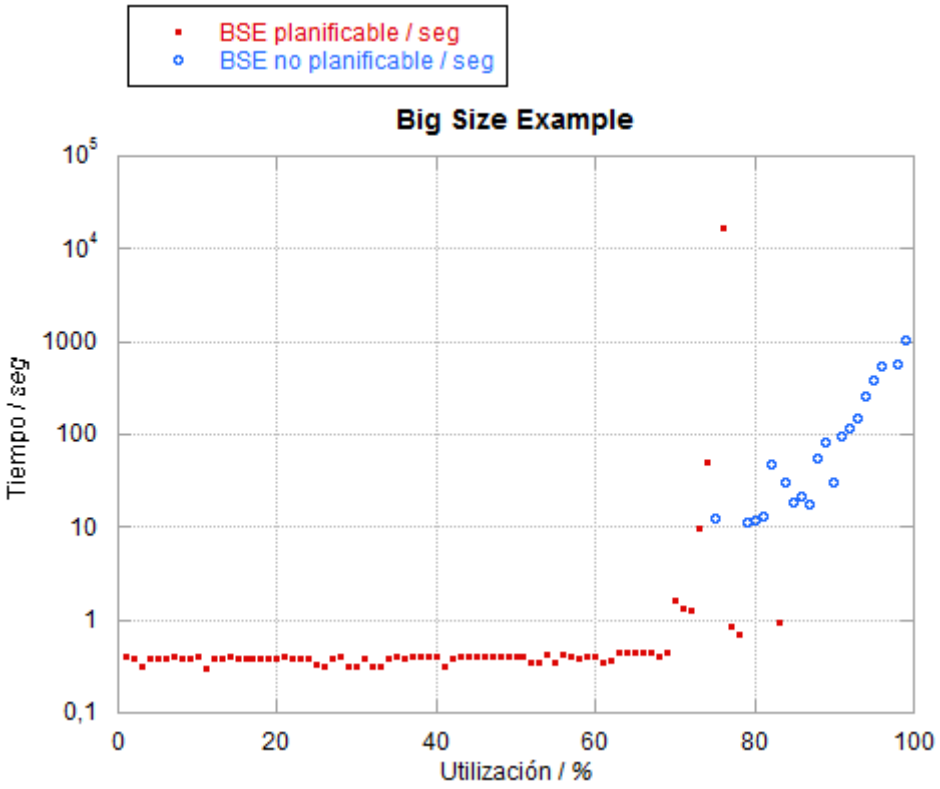
Sin embargo, pese a ser 7 horas lo que se tardaría en analizar toda la serie de manera secuencial, en Calderón el tiempo de análisis necesario es similar al orden de magnitud del tiempo máximo, que es de hora y media para el correspondiente al 88%, no planificable. Durante el tiempo en que este sistema está siendo analizado y sus prioridades van variando, otros muchos sistemas llevan a cabo sus análisis más rápidamente.

En este caso se considera el valor máximo de la planificación al 86%, pese a que desde el 83 al 85% el análisis concluya siendo no planificable.

4.1.3.- Ejemplo grande (*BSE*)

El modelo original posee 8 procesadores y una utilización media de 9.78% (sus procesadores poseen las siguientes utilizaciones: 1.02%, 1.68%, 10.98%, 1.45%, 11.02%, 0.45%, 50.73%, 0.88%).

El análisis de toda la serie (el 17 de febrero del 2011 a las 16:53) obtuvo los siguientes resultados:



Gráfica 3. Representación de los tiempos de análisis de cada sistema en escala logarítmica. De nuevo existen puntos planificables tras algunos valores no planificables, aunque esta vez lo más destacable es el hecho de encontrar un valor planificable como el que más tiempo requiere (76%, con más de 4 horas de análisis). Además no se ha representado el punto correspondiente a la utilización del 97%, ya que después de 100 horas de ejecución seguía trabajando (se comprobó varias veces que este caso tomaba muchísimo tiempo en ser analizado) y se eliminó de la cola de trabajo.

Además del caso del 76%, que sí es planificable y que es el que más tiempo requiere, existe otro caso planificable que requiere un cantidad de tiempo similar a los casos no planificables. Este caso es el correspondiente a la utilización del 74% con un tiempo de 49 segundos, mientras que el caso intermedio entre estos dos casos, el del 75%, no es planificable con un tiempo de 12 segundos, lo que quiere decir que realiza el análisis y las 180 asignaciones de prioridades en un tiempo inferior al necesario al análisis del 74%, y muy inferior a las 4 horas del 76%.

	Suma de tiempos / *seg*	Media del tiempo / *seg*	Desviación estándar / *seg*	Tiempo máximo / *seg*
Sistemas planificables (78)	16757.01	200	1900	16664.70
Sistemas no planificables (20)	3488.37	170	180	1034.95
Todos los sistemas (98)	20245.38	200	1700	16664.70

Tabla 5. Sumatorio de los tiempos de los trabajos según su planificación y del total del ejemplo grande (*BSE*). Además de los valores medios, desviaciones estándar y tiempos máximos.

El tiempo total empleado es de más de 5 horas y media, aunque no se tiene en cuenta que el 97% estuviese trabajando durante 100 horas sin resultado. Aunque es destacable que el 75% ha sido declarado no planificable con el mismo número o mayor de iteraciones de análisis que el 76%, y aún así ha tardado del orden de 1000 veces menos en ser analizado, hay que tener en cuenta que el tiempo del análisis crece a medida que aumenta la carga del sistema, acentuándose especialmente a medida que se alcanza el límite de planificabilidad.

Por último, la máxima utilización planificable se localiza en el 83%, después de haber calificado del 79% al 82% como no planificables.

4.2.- Comparación de rendimientos entre grupos de procesadores de Calderón

Además de usarse Calderón para ejecutar MAST se ha usado MAST para conocer mejor algunas características de Calderón. En este caso se ha usado para conocer la diferencia de rendimientos entre los distintos grupos de procesadores de Calderón (Ver Anexo I). Hay que tener en cuenta que la comparación se realizará solamente entre los grupos de ejecuciones secuenciales (dado que MAST está programado de forma secuencial) y que las ejecuciones se llevaron a cabo antes de febrero del 2011 (cuando se añadieron nuevos procesadores al *cluster* y se cambió la arquitectura de éste).

Para comprobar la diferencia de velocidades de los distintos grupos de procesadores se analizó un sistema cualquiera de MAST cuya solución no es relevante, aunque sí lo es el tiempo empleado en resolverlo.

Para llevar a cabo la comparación se analizaron cinco sistemas distintos con la misma arquitectura y se representaron los valores medios obtenidos. En cada grupo se analizaron exactamente estos cinco sistemas (los mismos modelos MAST) con la misma herramienta y sin ninguna opción adicional más que el análisis, por tanto las condiciones son exactamente las mismas para cada grupo de procesadores.

Para este proyecto se ha ejecutado la mayoría de los trabajos en el grupo 2 a sabiendas de que es uno de los más lentos y por tanto el que menos demanda tiene entre los usuarios (encontrando muchas veces este grupo libre para ser usado sin esperas) pese al contrapunto que supone trabajar con el grupo más lento, aunque esto es en muchas ocasiones irrelevante (las ejecuciones se solventan como mucho en escasos minutos y en igualdad de condiciones para cada trabajo). Además de que como se puede apreciar en el Anexo I, es uno de los grupos con más procesadores después del grupo 3.

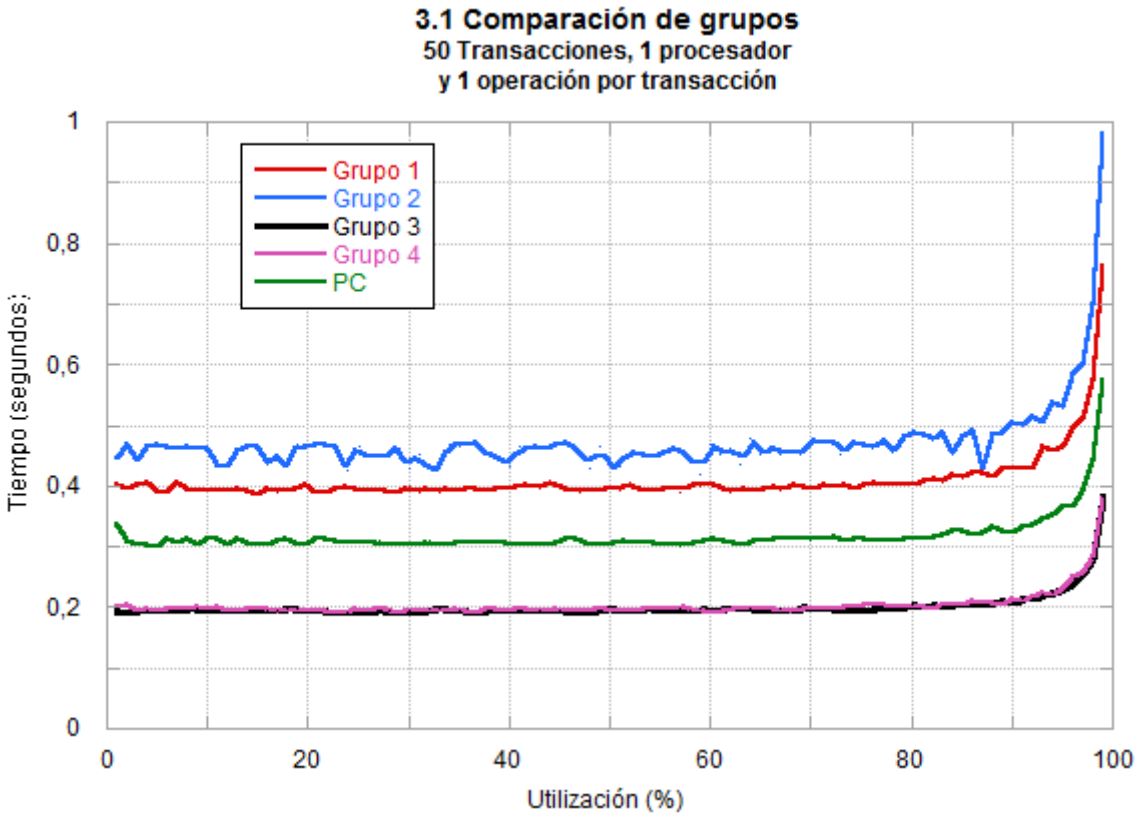
Los cinco sistemas con la misma arquitectura son los que se definen en la siguiente tabla:

Grupo	Momento ejecutado	Herram.	Tran.	Proc.	Op./Tran. y Proc.	Relación de tiempos de peor caso máximo / mínimo
g1	13:55 (31- Ene)	*Offset based*	50	1	1	9
g2	13:40 (31- Ene)					
g3	14:11 (31- Ene)					
g4 / gpu	14:45 (31- Ene)					

Tabla 6. Los cinco análisis para comparar los 4 grupos de procesadores para cálculos secuenciales se llevaron a cabo con sistemas de la arquitectura descrita. La herramienta elegida fue *offset based*, los resultados de máxima utilización planificable, aunque no es realmente relevante, es para cada uno de los 5 sistemas 78%, 77%, 78%, 81% y 78%. Aunque lo realmente importante es que el sistema sea idéntico para cada grupo y así poder comparar los tiempos necesarios para el análisis en igualdad de condiciones.

Por último hay que añadir que el autor ejecutó en su propio computador el mismo trabajo, comparándolo a las medidas obtenidas en Calderón. La diferencia más relevante de ejecución entre una máquina y la otra es que en Calderón todos los trabajos se calcularon casi totalmente de manera simultánea, mientras que en este computador cada trabajo fue analizado en serie.

Las especificaciones de este computador (PC) son: Modelo, *Dell Vostro 1510*; Procesador, *Intel(R) Core(TM)2 Duo CPU T8100 – 2.10 GHz 2.10GHz*; Memoria RAM, *2,00 GB*; Sistema operativo, *Linux Ubuntu 9.10 de 32 bits*, instalado sobre *Windows 7 Profesional de 32 bits* utilizando el *software Wubi 9.10*.



Gráfica 4. Comparación entre los grupos de procesadores con el análisis de modelos generados con la arquitectura de la tabla 6. La gráfica refleja que el grupo 2 es claramente el más lento, siendo los dos más veloces el grupo 3 y el grupo 4 (casi idénticos y superpuestos). Cada punto de la gráfica representa el tiempo medio tomado por cada 5 procesadores al analizar cada uno un sistema de utilización determinada. Por tanto la velocidad de los procesadores está medida de manera individual para cada procesador.

El grupo 1 tardó en torno al doble de lo que tardaron el grupo 3 o el grupo 4 de media (unos 0.20 segundos de estos grupos frente a los 0.41 segundos de media del grupo 1). El grupo 2 ejecutó la serie de sistemas con una media de 0.47 segundos siendo el más lento. La línea que se corresponde al tiempo de ejecución en el computador del autor (PC en la leyenda) es la línea verde, situándose en una posición intermedia, siendo la velocidad del procesador de este computador un caso intermedio entre los dos grupos más veloces y los dos más lentos de Calderón para trabajos secuenciales. El tiempo medio invertido para esta ejecución es de 0.32 segundos por trabajo.

Grupo	Suma de tiempos / *seg*	Media / *seg*	Desviación Std. / *seg*	Error / *%*	Máximo / *seg*
g1	202,67	0,41	±0,05	11,28	0,89
g2	234,02	0,47	±0,08	16,26	1,32
g3	99,93	0,20	±0,02	11,66	0,44
g4	100,73	0,20	±0,02	11,60	0,43
PC	157,29	0,32	±0,03	10,76	0,67

Tabla 7. Valores estadísticos extraídos de los análisis llevados a cabo a partir de la arquitectura de

la tabla 7. La suma de tiempos se corresponde a la suma de los 495 tiempos totales, 5 series de 99 sistemas. El error se refiere al error relativo, definido como la relación entre la desviación estándar y la media.

De esta tabla se extrae que al tomar un valor medio de las casi 500 ejecuciones (99 por cada sistema analizado, de los que cada uno de los 5 sistemas generan un valor medio representado en la gráfica 4) se puede observar cuál de los 4 procesadores tiene un ejecución más lenta, como ya se ha visto gráficamente, el grupo 2, siendo el grupo 3 y 4 casi idénticos, los más veloces de la comparativa.

El error, procedente de la desviación estándar, refleja además que el grupo 2, como se ve en la gráfica 4, sufre bastantes fluctuaciones, los puntos no está bien alineados debido a que los procesadores no son muy homogéneos, pese a haber representado la media de 5 ejecuciones distintas. Este error también es fruto del crecimiento que sufre la gráfica para los últimos valores de la utilización (donde existe un pico que rompe la horizontalidad), pero este mismo pico existe en las ejecuciones en los otros grupos y no eleva el error hasta el 16%, sino que toma un valor del 11% muy similar en los otros 3 casos en Calderón, incluyendo el del grupo 1, que pese a tener el pico el doble de grande que los grupos 3 y 4 mantiene el error en dicha misma cota.

El error del PC es menor del 11% y es menor que el resto por el hecho de que al ser el mismo procesador el que ejecuta cada trabajo y en exactamente las mismas condiciones, no fluctúa tanto de un análisis al siguiente.

Hay que añadir un importante punto en este análisis. Si se considera el momento en que el primer análisis en cada grupo comenzó, y el momento en que el último análisis terminó, se puede de esta manera calcular el tiempo total de ejecución de los 99 trabajos, en el caso del PC es evidentemente la suma de todos los análisis, pero para cualquier grupo de procesadores de Calderón. De esta forma se puede medir la velocidad del grupo como un conjunto, bastante más práctico a la hora de paralelizar ejecuciones, como es el caso del proyecto.

Para medir el tiempo que ha tardado cada grupo de procesadores en realizar el análisis de una serie de 99 sistemas, se anota la hora en que los archivos de salida de cada análisis fueron creados. Estas horas de creación se pueden observar en las propiedades de los archivos, en la fecha de modificación (no fecha de creación, ya que la fecha de creación se corresponde a la hora en que dicho archivo fue copiado del *cluster* al ordenador del usuario, y la fecha de modificación se corresponde a la última vez en que dicho archivo recibió información nueva y se guardó en él, que si no ha sido modificado por terceros, ésta debe ser en la creación del fichero).

De este modo, el método seguido para medir la duración total es una aproximación en la que se mide la diferencia de la creación del fichero de salida del último en terminar con la creación del fichero de salida del primero en terminar, despreciando las duraciones de los análisis (ya que éstos son más pequeños que el error de medida de las horas de creación, que son del orden del segundo):

Grupo de procesadores	Serie	Hora inicio	Hora final	Tiempo total / *seg*	Tiempo medio /*seg*	Desviación estándar / *seg*
g1	1ª	14:06:54	14:06:55	1	0.4	0.5
	2ª	13:56:07	13:56:07	0		
	3ª	14:06:56	14:06:57	1		
	4ª	14:06:56	14:06:56	0		
	5ª	14:06:58	14:06:58	0		
g2	1ª	13:44:31	13:44:36	5	1	2
	2ª	13:44:36	13:44:37	1		
	3ª	13:44:37	13:44:37	0		
	4ª	13:44:37	13:44:37	0		
	5ª	13:44:37	13:44:37	0		
g3	1ª	14:34:09	14:34:09	0	0	0
	2ª	14:34:10	14:34:10	0		
	3ª	14:34:10	14:34:10	0		
	4ª	14:34:10	14:34:10	0		
	5ª	14:34:10	14:34:10	0		
g4	1ª	14:50:41	14:50:42	1	1.6	0.5
	2ª	14:50:44	14:50:46	2		
	3ª	14:50:48	14:50:50	2		
	4ª	14:50:52	14:50:54	2		
	5ª	14:50:55	14:50:56	1		
PC	1ª	11:58:25	11:58:58	33	33.0	0.7
	2ª	12:00:36	12:01:09	33		
	3ª	12:03:43	12:04:16	33		
	4ª	12:04:54	12:05:28	32		
	5ª	12:06:31	12:07:05	34		

Tabla 8. Horas de creación de los archivos de salida *.TIME* de los análisis correspondientes a la comparación de grupos de procesadores, acompañado de los tiempos calculados como la diferencia de las horas final e inicial. El error de la medida de la hora es de ±1 segundo, por lo que la diferencia de tiempos tiene un error que será el peor caso entre 2 segundos de la propagación de errores o la desviación estándar (se da el caso de que todos los valores son menores o iguales a la propagación de errores).

Como se puede observar de la tabla 8, el grupo 4, pese a ser más rápido (a nivel individual de los procesadores) que el grupo 1 y el 2, es el que más tiempo ha precisado para analizar toda la serie de sistemas. Esto se debe a que el grupo 4 posee un reducido número de procesadores (8), mientras que el grupo 2 dispone de muchos más (92), aunque más lentos que los del grupo 4. Al ser tan pocos procesadores en el grupo 4, sólo puede llevar a cabo 8 análisis simultáneos, de 99, dejando muchos en la cola esperando a que se libere uno de los 8 procesadores.

Por otro lado, trabajando con el grupo 3 se ha tardado menos de un segundo en ejecutar la serie de análisis, es decir, ha tardado cero segundos (con su correspondiente error de dos segundos). Si se tiene en cuenta que cada análisis ha tardado en torno a una quinta parte del segundo, que es la

precisión del método, y que dado el gran número de procesadores de este grupo (256), la mayoría (si no todos) de los análisis se ejecutaron simultáneamente, con lo que parece bastante razonable que toda la serie sea ejecutada en el mismo segundo del reloj.

La conclusión más relevante de la tabla 8 se corresponde con uno de los objetivos del proyecto, que es el de reducir de manera drástica los tiempos necesarios para ejecutar grandes cantidades de análisis. Como se puede observar, el análisis de toda la serie, en Calderón, puede llevar en torno a uno o dos segundos, mientras que en el PC hicieron falta 33 segundos. Si se complicase mucho el análisis o se incrementase mucho el número de análisis, tomando por ejemplo 110000 análisis en vez de los 99, como en la bibliografía **[6]**, el tiempo necesario en un sólo computador sería de unas 10 horas, mientras que en Calderón usando un sólo grupo (no distribuyéndolo, que sería lo más sensato) tomaría entre media hora y 7 minutos.

Capítulo 5
Conclusiones

5.1.- Conclusiones generales

En primer lugar, centrando la atención en el principal objetivo de este proyecto (que es la de implementar los mecanismos necesarios para poder ejecutar de manera paralela MAST y todas sus herramientas de análisis y optimización), cabe destacar lo obtenido en el apartado 4.2, en donde se observa que paralelizando 99 análisis de MAST supone un ahorro de tiempo de casi el 95% (como poco). Hay que tener en cuenta que para paralelizaciones más extremas (con un número mucho mayor de análisis) las diferencias es muy probable que se incrementen aún más.

Es curioso además el caso de dos análisis en concreto del apartado 4.1.3, donde el 76% del *BSE* y el 97% tomaron mucho tiempo para ser analizados (de hecho el 97% fue abortado al alcanzar las 100 horas de análisis). Si esto ocurriese en una serie de ejecuciones en un monoprocesador sería muy contraproducente, ya que toda la cola de trabajos se vería retrasada y bloqueada por estos trabajos. Se puede comparar a modo de símil con las luces de Navidad. En las luces de Navidad las bombillas de colores se conectan en un circuito eléctrico en paralelo, de manera que si una bombilla se funde el resto de bombillas sigan funcionando. Si se colocasen en serie todas las bombillas se apagarían al no circular corriente por la bombilla fundida. Del mismo modo es mucho más eficiente disponer de otros muchos procesadores que ejecuten todos los análisis “normales” mientras algún procesador puede encontrarse con algún análisis que requiera de más tiempo, sin que éste suponga un estorbo para el resto de trabajos de la cola.

Por tanto, la paralelización no sólo supone un ahorro de tiempo y de recursos computacionales, sino que además supone una liberación ante esos problemas que pueden bloquear largas listas de análisis.

La pega en el uso de estos *cluster*s es la que supone trabajar de manera remota y con el uso de *scripts* (cuando esto no es algo intuitivo al ejecutar un programa en un computador común). Pero estas pegas suponen también otros beneficios y ventajas. Al poder trabajar de manera remota, normalmente sin un entorno visual agradable (sólo el uso de la *shell* de Unix), el usuario puede trabajar desde cualquier computador con acceso a Internet y con Unix, sin necesidad de tener instalado un determinado *software* o que disponga de un *hardware* suficientemente potente para poder ejecutarlo; con conectarse al *cluster* con un terminal es suficiente (si el material de trabajo está ya en el *cluster*) para trabajar desde él. Además, el uso de los *scripts* permite controlar la ejecución (elegir grupo de procesadores, definir los tipos de archivos de salida, de errores, etc) y es totalmente necesario a la hora de organizar tantos trabajos como ocurre a la hora de paralelizar. Y más concretamente al existir una cola de gestión de trabajos para tantos usuarios, ya que es necesaria la correcta organización de las ejecuciones para no comprometer las capacidades del *cluster.*

Es precisamente a la hora de aprender a usar el *cluster* donde más dificultades se encontraron durante este proyecto, especialmente con el diseño de los *scripts*, muchas veces mal diseñados y de los que se obtenía errores en vez de ejecuciones correctas. Destáquese también la experiencia necesaria en Unix para poder usar correctamente la *shell*, el intercambio de archivos entre un servidor (el *cluster*) y el computador cliente (desde el que se conecta el usuario al servidor por medio de Internet), la gestión de la cola de trabajos de Calderón, etc. Que son necesarios a la hora de funcionar correctamente en el *cluster*.

Además de la experiencia desarrollada en torno al *cluster*, es necesario adquirir ciertos conocimientos sobre los sistemas de Tiempo Real para este proyecto, especialmente orientado al uso del *software* MAST. Desde conocer los elementos que forman estos sistemas, entender el funcionamiento de los mismos, las diferentes maneras de analizarlos, sus variables, etc. Todos esto es necesario a la hora de desarrollar herramientas *software* que generen modelos MAST (sistemas de Tiempo Real al fin y al cabo) y a la hora de estudiarlos.

5.2.- Conclusiones sobre el uso de grupos de procesadores de Calderón

En Calderón existen diferencias interesantes entre los distintos grupos de procesadores para trabajos secuenciales. Se realizó una prueba en el apartado 4.2 para diferenciar la velocidad de los procesadores en los distintos grupos (ver tabla 7), obteniéndose que la velocidad de los grupos queda diferenciada de la siguiente forma:

$$V_{g2} < V_{g1} < V_{g3} = V_{g4}$$

Sin embargo al considerarlos colectivamente, en el apartado 4.2 en el tabla 8, el orden de las velocidades sería el siguiente:

$$V_{g4} < V_{g2} < V_{g1} < V_{g3}$$

En este caso, ello es debido especialmente al número de procesadores que forman dicho grupo, no sólo a la velocidad de los mismos. Se puede observar la cantidad de éstos en la siguiente tabla:

Grupo	Num. Nodos	Num. Procesadores por nodo	Num. Total procesadores
g1	9	2	18
g2	23	4	92
g3	32	8	256
g4	1	8	8

tabla 9. Relación del número de procesadores en total en cada grupo, desglosando el número de nodos de procesadores de cada grupo (entendiendo como *nodo* "una agrupación de procesadores interconectados física y computacionalmente"), así como la cantidad de procesadores en cada nodo.

Aunque exista esta diferencia entre los grupos (recuérdese que se está hablando de grupos de ejecuciones secuenciales, no se ha comprobado con los grupos de compilación o de trabajos

paralelizables), ha de tenerse en cuenta que existen otras muchas diferencias, como la respuesta de los procesadores de un grupo para trabajos más complejos o la arquitectura del grupo.

Gracias a estos resultados se extraen las siguientes conclusiones:

1. Los distintos grupos de procesadores están bien diferenciados en cuanto a la calidad de sus procesadores, salvando las similitudes entre el grupo 3 y 4.
2. El grupo 3 posee los procesadores más rápidos además de un gran número de ellos, es por tanto normal que sea el grupo con más demanda entre sus usuarios.
3. El grupo 4 pese a tener los mismos procesadores que el grupo 3 en cuanto a velocidad individual de éstos, su reducido número los convierte en una clara última opción para trabajos de paralelización, como han sido los de este proyecto, y más si además alguno de sus procesadores está siendo ya usado por otros trabajos ajenos al usuario.
4. El grupo 2 es a nivel individual de sus procesadores el más lento, y no sólo eso, sino que la poca homogenización de estos procesadores (se trata de procesadores de doble núcleo con frecuencias de 1.8 y 2.2 GHz) lo convierte en una mala opción a priori, pero posee un número elevado (aunque no tanto como el grupo 3) de procesadores, lo que unido a su poca demanda entre los usuarios, facilita su uso para la paralelización de trabajos.
5. El grupo 1 queda por tanto, con unos procesadores de velocidad intermedia, y un número reducido de éstos como una opción mediocre para ejecutar trabajos individuales, aunque no adecuada para paralelizar trabajos.

Hay que destacar que en febrero de 2011 se añadieron nuevos procesadores al *cluster*, variando enormemente la arquitectura de éste, creándose varios grupos de procesadores nuevos. Además algunos procesadores han sido dados de baja en el *cluster*, como el `g1` al completo, dada su escasa potencia de cálculo frente a su elevado consumo energético (según medidas del administrador del *cluster*, la corriente medida en uno de sus procesadores para cargas del 0% y del 100% de trabajo es de 0.84 A y 1.05 A correspondientemente, frente a los 0.44 A al 0% de carga y 0.84 A al 100%, medidos en uno de los nuevos procesadores instalados).

En el Anexo I se muestra la disposición del hardware en Calderón antes y después de este cambio (hay que tener en cuenta que la comparación de grupos se llevó a cabo antes de dichas reformas).

Además de los grupos nuevos detallados en el Anexo I se han creado dos grupos personalizados con procesadores de otros grupos para ser usados en trabajos que requieran de la máxima potencia de cálculo. El primer grupo es el denominado ”`gfast`” que cuenta con los nuevos procesadores (más rápidos que los usados hasta ahora), y el segundo grupo se llama “`ghuge`” en el que se aglutinan los que mayor memoria RAM tienen instalada, es decir, a cada procesador en Calderón se le asignan 2 Gb de memoria RAM, en el caso de este grupo, sus procesadores tienen instalados 4 Gb de memoria RAM cada uno, y al estar formado por 12 procesadores, el grupo posee 48 GB.

Bibliografía

[1] Grupo de Arquitectura y Tecnología de Computadores (ATC), manual para el uso de Calderón *Manual_SGE_Calderon_V9.pdf.*

[2] *Documentación de Mast* (http://mast.unican.es/#docs) del Grupo de Computadores y Tiempo Real (CTR).

[3] Apuntes de la asignatura *Sistemas de Tiempo Real* de la carrera de Ingeniería Informática de esta Universidad. J. Carlos Palencia Gutiérrez y Héctor Pérez Tijero.

[4] "*Análisis de planificabilidad de sistemas distribuidos de tiempo real basados en prioridades fijas*" J. C. Palencia Gutiérrez y M. González Harbour (Director). Tesis doctoral, Universidad de Cantabria, 1999.

[5] "*Meta-planificación de trabajos Grid en múltiples clusters con acceso local*" José Carlos Blanco Real , Trabajo Fin de Máster.

[6] "*Optimized Deadline Assignment and Schedulability Analysis for Distributed Real-Time Systems with Local EDF Scheduling*" Juan M. Rivas, J. Javier Gutiérrez, J. Carlos Palencia, y Michael González Harbour, 8th International Conference on Embedded Systems and Applications, ESA'2010, Las Vegas (Nevada, EE.UU.), Julio 2010.

[7] "*Optimized Priority Assignment for Tasks and Messages in Distributed Real-Time Systems* " J.J. Gutiérrez García and Michael González Harbour. Proceedings of 3rd Workshop on Parallel and Distributed Real-Time Systems, IEEE Computer Society Press, pp. 124-132, April 1995.

[8] *API de Java* (http://download.oracle.com/javase/1.5.0/docs/api/) Java ™ 2 Platform Standard Ed. 5.0.

[9] *Guía Ubuntu* online (http://www.guia-ubuntu.org/).

[10] John R. Hubbard, Profesor de Matemáticas y Ciencia Computacional en la Universidad de Richmond. *Schaum's outline of Theory and Problems of Programming with C++*, McGraw Hill, 2000, segunda edición.

[11] Paquete de programas para Java *fundamentos* y su documentación, que es un paquete modificado por Michael González Harbour, profesor y miembro del grupo de Computadores y Tiempo Real (CTR), del paquete *javagently* para Java, creado por Judith Bishop.

Agradecimientos

A José Ángel Herrero Velasco, administrador del *cluster* Calderón y miembro del grupo de Arquitectura y Tecnología de Computadores (ATC), por su gran ayuda a la hora de aprender a usar Calderón y del uso de *scripts* en Unix.

A Pablo García Fernández, miembro del departamento de Ciencias de la Tierra y la Materia Condensada (CITIMAC), del grupo de Física Computacional de Materiales, por su ayuda en la toma de contacto con Calderón.

A Alberto Barba Rueda, Cristina Echevarría Bonet y Borja García Cueto, compañeros de la facultad que me han ayudado con dudas puntuales en Java, Calderón y Unix.

Y a mis dos directores de proyecto, J. Carlos Palencia Gutiérrez y J. Javier Gutiérrez García.

Anexo I
Nodos del *cluster*

Los siguientes nodos se corresponden a la organización dispuesta antes de las reformas llevadas a cabo en febrero de 2011:

Nodos de compilación:

Grupo	Nombre	Descripción	Comando
Grupo 1	compilacion-0	Nodo con 4 procesadores AMD Opteron 275 a 2.2 Ghz y 8 GB de memoria principal cada uno. Integra todo tipo de herramientas y librerías para la compilación de fuentes, tanto de GNU como de PathScale.	`-l comp-g1` `-l comp-ps`
Grupo 2	compilación-1	Nodo con 4 procesadores AMD Opteron 275 a 2.2 GHz y 8 GB de memoria principal cada uno. Integra todo tipo de herramientas y librerías para la compilación de fuentes de GNU.	`-l comp-g2`
Grupo3	compute-gpu	Nodo con 8 procesadores Intel Xeon 5472 a 3 GHz y 16 GB de memoria principal cada uno. Equipado como se describió antes con 2 tarjetas GPU Nvidia Tesla C1060 GPU. Integra herramientas y librerías GNU adecuadas para la compilación de fuentes contra la tarjetas gráficas.	`-l comp-gpu`

Nodos de cálculo de trabajos secuenciales:

Grupo	Nombre y nodos	Descripción	Comandos

Grupo 1	`g1` - c0-0 a c0-8	Grupo formado por 9 nodos con 2 procesadores AMD Opteron 248 a 2.2 Ghz y 4 GB de memoria principal cada uno.	`-l long-g1`
			`-l short-g1`
			`-l inter-g1`

Grupo 2	`g2` - c0-9 a c1-31	Grupo formado por 23 nodos con 4 procesadores AMD Opteron 265/275 a 1.8/2.2 Ghz y 8 GB de memoria principal cada uno.	`-l long-g2`
			`-l short-g2`
			`-l inter-g2`

Grupo 3	`g3` - c2-32 a c3-63	Grupo formado por 32 nodos con 8 procesadores Intel Xeon 5472 a 3 GHz y 16/32 GB de memoria principal cada uno.	`-l long-g3`
			`-l short-g3`
			`-l inter-g3`

Grupo GPU	`g4` - cgpu-0	Grupo formado por 1 nodo con 8 procesadores Intel Xeon 5472 a 3 GHz y 16 GB de memoria principal cada uno. Equipado como se describió antes con 2 tarjetas GPU Nvidia Tesla C1060 GPU.	`-l long-gpu`
			`-l short-gpu`
			`-l inter-gpu`

Nodos de cálculo de trabajos paralelos:

Grupo	Nodos	Descripción	Comandos

Grupo ethernet	c0-0 a c3-63	Todos los nodos del *cluster*.	`-l long-parallel-eth`
			`-l short-parallel-eth`
			`-l inter-parallel-eth`

Grupo myrinet	c0-9 a c1-31	Grupo formado por 16 nodos con 4 procesadores AMD Opteron 265 a 1.8 GHz y 8 GB de memoria principal cada uno.	`-l long-parallel-myr`
			`-l short-parallel-myr`
			`-l inter-parallel-myr`

Grupo infiniband	c2-32 a c3-63	Grupo formado por 24 nodos con 8 procesadores Intel Xeon 5472 a 3 GHz y 16 GB de memoria principal cada uno.	`-l long-parallel-inf`
			`-l short-parallel-inf`
			`-l inter-parallel-inf`

Grupo GPU	cgpu-0	Grupo formado por 1 nodo con 8 procesadores Intel Xeon 5472 a 3 GHz y 16 GB de memoria principal cada uno. Equipado como se describió antes con 2 tarjetas GPU Nvidia Tesla C1060 GPU.	`-l long-parallel-gpu`
			`-l short-parallel-gpu`
			`-l inter-parallel-gpu`

A partir de aquí se muestra la organización de los nodos después de la actualización de febrero de 2011:

Nodos de cálculo de trabajos secuenciales:

Grupo	Nombre	Descripción	Comandos

Grupo 1	`g1`	Grupo formado por 9 nodos con 2 procesadores (2 cores) AMD Opteron 248 a 2.2 GHz y 4 GB de memoria principal cada uno.	`-l long-g1`
			`-l short-g1`
			`-l inter-g1`

Grupo 2	`g2`	Grupo formado por 23 nodos con 2 procesadores (4 cores) AMD Opteron 265/275 a 1.8/2.2 GHz y 8 GB de memoria principal cada uno.	`-l long-g2`
			`-l short-g2`
			`-l inter-g2`

Grupo 3	`g3`	Grupo formado por 16 nodos con 2 procesadores (8 cores) Intel Xeon 5472 a 3 GHz y 16 GB de memoria principal cada uno.	`-l long-g3`
			`-l short-g3`
			`-l inter-g3`

Grupo 4	`g4`	Grupo formado por 16 nodos con 2 procesadores (8 cores) Intel Xeon 5472 a 3 GHz y 16/32 GB de memoria principal cada uno.	`-l long-g4`
			`-l short-g4`
			`-l inter-g4`

Grupo 5	`g5`	Grupo formado por 4 nodos con 2 procesadores (8 cores) Intel Xeon 5550 a 2,67 GHz y 32 GB de memoria principal cada uno.	`-l long-g5`
			`-l short-g5`
			`-l inter-g5`

Grupo 6	`g6`	Grupo formado por 20 nodos con procesadores (12 cores) Intel Xeon 5650 a 2,67 GHz y 24/48 GB de memoria principal cada uno.	`-l long-g6`
			`-l short-g6`
			`-l inter-g6`

Grupo GPU	`g4` - cgpu-0	Grupo formado por 1 nodo con 8 procesadores Intel Xeon 5472 a 3 GHz y 16 GB de memoria principal cada uno. Equipado como se describió antes con 2 tarjetas GPU Nvidia Tesla C1060 GPU.	`-l long-gpu`
			`-l short-gpu`
			`-l inter-gpu`

El apartado “Comandos” expone la forma en que deben ser invocados los nodos al usar por ejemplo el comando `qrsh`.

Anexo II
Lista de archivos fuente del *software* y *script* de compilación de los programas del autor

Listado de archivos y carpetas:

Como parte del material de este proyecto se entregan los siguientes archivos fuentes (de formato .java), un *script* y carpetas (con sus correspondientes archivos) listos para ser compilados y generar las herramientas *software* necesarias:

- BuscaSchedulable.java
- InputMast.java
- Lector.java
- Operation.java
- Principal.java
- Processor.java
- Red.java
- Scheduler.java
- SchedulingServer.java
- ScriptCalderon.java
- ScriptsUnix.java
- Tiempo.java
- Transaction.java
- Utilization.java
- compile.sh
- fundamentos/...
- ManifestBuscador/MANIFEST.MF
- ManifestCalderon/MANIFEST.MF
- ManifestLector/MANIFEST.MF
- ManifestMast/MANIFEST.MF
- ManifestPrincipal/MANIFEST.MF

Código de compilación:

A continuación se encuentran los comandos de compilación y generación de ejecutables de los programas hechos por el autor, este texto se encontrará en el *script compile.sh* junto a los archivos ya listados.

El código del *script* ejecuta varios comandos secuencialmente con el fin de obtener varios ejecu-

tables (Principal.jar, InputMast.jar, ScriptCalderon.java, LectorTiempos.jar y BuscaSchedulable.jar), el código es:

```
rm -r Ejecutables
mkdir Ejecutables

javac Principal.java
jar cmf ManifestPrincipal/MANIFEST.MF Ejecutables/Principal.jar
Principal.class InputMast.class Processor.class Red.class
Scheduler.class SchedulingServer.class Operation.class
Transaction.class Utilization.class ScriptCalderon.class
ScriptsUnix.class fundamentos

javac InputMast.java
jar cmf ManifestMast/MANIFEST.MF Ejecutables/InputMast.jar
InputMast.class Principal.class Processor.class Red.class
Scheduler.class SchedulingServer.class Operation.class
Transaction.class Utilization.class fundamentos

javac ScriptCalderon.java
jar cmf ManifestCalderon/MANIFEST.MF
Ejecutables/ScriptCalderon.jar ScriptCalderon.class
Principal.class fundamentos

javac Lector.java
jar cmf ManifestLector/MANIFEST.MF Ejecutables/LectorTiempos.jar
Lector.class Principal.class Tiempo.class ScriptCalderon.class

javac BuscaSchedulable.java
jar cmf ManifestBuscador/MANIFEST.MF
Ejecutables/BuscaSchedulable.jar BuscaSchedulable.class
Principal.class

rm *.class
```

Al ejecutar el *script*, éste debe encontrarse en el mismo directorio en que estén localizados todos los ficheros y carpetas anteriormente listados. Al terminar de ejecutarse, el *script* habrá creado una carpeta llamada “Ejecutables” donde se encontrarán los archivos binarios.