Entorno de desarrollo para clusters

Memòria del Projecte Fi de Carrera
d'Enginyeria en Informàtica
realitzat per
Javier Panadero Martínez
i dirigit per
Porfidio Hernádez Budé
Bellaterra 6 de Junio de 2008

Universitat Autònoma de Barcelona

etse
Escola Tècnica Superior d'Enginyeria

El sotasignat, Porfidio Hernández Budé

Professor/a de l'Escola Tècnica Superior d'Enginyeria de la UAB,

CERTIFICA:

Que el treball a què correspon aquesta memòria ha estat realitzat sota la seva direcció per en

I per tal que consti firma la present.

Signat: .....

Bellaterra, ...de.....de 200.....

Deseo expresar mi agradecimiento al Dr. Porfidio Hernández por su magistral dirección y constante dedicación a la supervisión del proyecto.

A Eduard Mallén por sus acertadas correcciones durante la elaboración de la memoria.

A Miguel Molina por su apoyo moral desde el principio de este proyecto.

A Roi, Tamara y Rebeca por haberlos tenido "abandonados" durante la elaboración de este proyecto.

Y como no, a toda mi familia.

Índice

Capítulo 1

Introducción

1.1 Presentación del problema

La simulación, junto con la teoría de modelos y el cómputo masivo, ha abierto una frontera a la posibilidad de abordar aplicaciones de la ciencia que hasta el momento permanecían vetadas (predicción de terremotos, descubrimiento del nacimiento de galaxias, simulación de reacciones nucleares, avances en genética, etc).
Esto se ha conseguido gracias a la utilización de entornos cluster, multicluster y grid como paradigma de resolución de problemas en cómputo de altas prestaciones.
Actualmente el uso de estos entornos se está empezando a introducir en aplicaciones comerciales (correo electrónico, servidores web, bases de datos, etc), lo que puede suponer el arranque definitivo de los entornos distribuidos de cómputo en la resolución de problemas de diversa índole.
Debido a la diversidad de aplicaciones existentes y dependiendo del tipo de aplicación a ejecutar, seleccionaremos un determinado tipo de cluster. La figura 1.1 muestra la taxonomía de los diferentes tipos de cluster en función del objetivo.

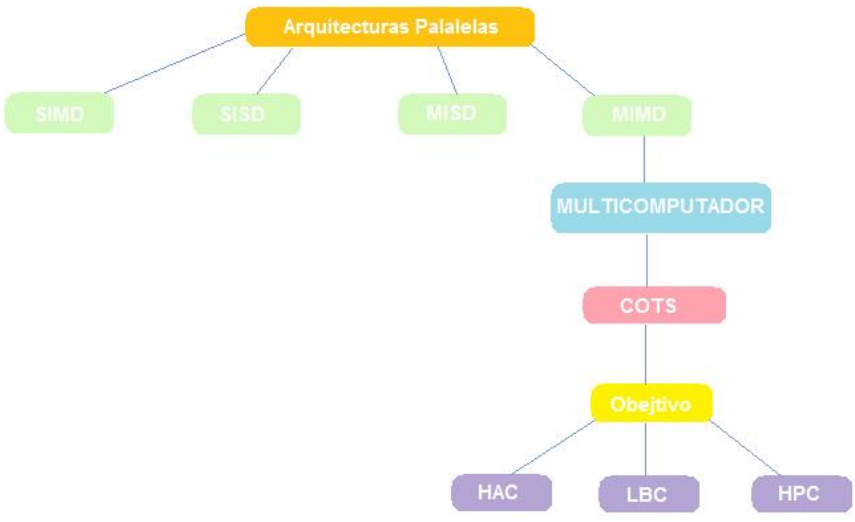


Figura 1.1: Taxonomía aplicada a las arquitecturas paralelas

Esta taxonomía arranca con los sistemas COTS (commodity of-the-shelf). Dichos sistemas se caracterizan por estar formados por ordenadores comerciales; obteniendo prestaciones similares a las que obtendríamos con sistemas MPP (Massively Parallel Processor). Los sistemas COTS se han convertido en una de las plataformas más atractivas y utilizadas para funcionar como supercomputadores. Es por esa razón que se necesita mejorar estas plataformas para ofrecer mejores prestaciones y escalabilidad, a parte de mejorar servicios tales como la disponibilidad, fiabilidad, calidad de servicio, bajo consumo, etc.
Centrando el estudio en estos sistemas, existen diferentes clasificaciones en función de sus objetivos. Cabe comentar que la taxonomía mostrada en la figura anterior (Figura 1) no es única. Puede variar en función de la manera de clasificar los sistemas multicomputador. Por ejemplo en lugar de clasificarlos en función del objetivo, como se ha hecho en este estudio, se podría haber hecho en función de la utilización de recursos.
Atendiendo a la figura 1.1, observamos que los sistemas COTS se pueden dividir en otros tipos de sistemas, como los HAC (High Availability Cluster), que engloba a los sistemas de alta disponibilidad y son usados en servidores webs y bases de datos. Otro tipo de sistemas que encontramos dentro de los COTS son los LBC (Load Balancing Cluster), los cuales engloban a su vez a los sistemas de balanceo de carga. Por último encontramos los sistemas HPC (High Performance Computing). En este estudio nos centraremos en este último tipo de sistemas.

El modo de acceso, independientemente del tipo de sistema utilizado, se puede llevar a cabo de dos formas distintas, vía consola remota a través de SSH o mediante la utilización de entornos gráficos (GUI). La forma empleada para comunicarnos con el cluster no interfiere en la llegada de los trabajos enviados al scheduler, donde se van posicionando en la cola de ejecución a la espera de ser enviados a los nodos de cómputo para su ejecución.
La figura 1.2 muestra uno de los posibles esquemas de comunicación con el cluster.
El scheduler es el encargado tanto de distribuir los trabajos en la cola de ejecución como de seleccionar los trabajos que se ejecutarán en cada momento en los nodos de cómputo. Dicho planificador se puede ajustar según las necesidades del administrador del cluster, dependiendo de las aplicaciones que necesitemos lanzar al cluster en cada momento o de la carga de trabajo, mediante las diferentes políticas de planificación.

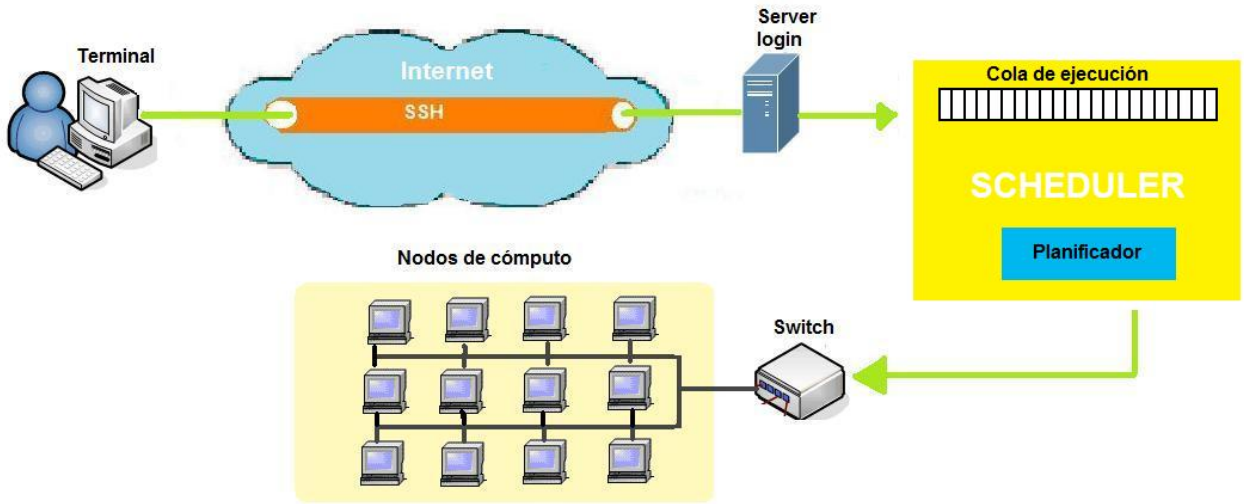


Figura 1.2: Esquema de comunicación con el cluster mediante SSH

Para la correcta ejecución de un trabajo, independientemente de su índole o condición, es necesario conseguir un entorno de trabajo idóneo que propicie un desarrollo óptimo del trabajo. Un trabajo cuya ejecución se demorara más de lo previsto inicialmente, provocaría que el resultado fuera de poca utilidad o en el peor de los casos no válido. Un ejemplo típico y que ilustra claramente la explicación sería la previsión meteorológica, la cual, evidentemente hay qué obtener antes de la fecha a predecir, de lo contrario sería de poca utilidad y habríamos malgastado recursos en vano, con el consiguiente gasto económico que eso supone. Para intentar solventar el problema anterior, uno de los objetivos a cumplir es utilizar una técnica óptima de planificación de aplicaciones paralelas, para intentar aprovechar los recursos al máximo.
Las técnicas de planificación paralelas, a diferencia de entornos clásicos monoprocesador, donde su principal objetivo es decidir que proceso toma el control de la CPU, se encargan de decidir cuando un proceso se ejecutará (planificación temporal) y donde será ejecutado dicho proceso (planificación espacial). En la figura 1.3, mostramos una primera clasificación muy general, que divide la planificación de aplicaciones paralelas en dos grandes grupos, planificación espacial y planificación temporal.

Figura 1.3: Planificación de aplicaciones paralelas

Las técnicas de planificación temporal las podemos dividir en tres grandes grupos: planificación temporal explícita, planificación temporal implícita y planificación temporal híbrida.

- *Planificación temporal explícita:* Está basada en el sistema de Gang Scheduling, también conocida como Task Forces o coscheduling. Dicha planificación consiste en el cambio global de contexto simultáneo a lo largo de toda la máquina paralela. Esta técnica es más apropiada para entornos dedicados que distribuidos.

- *Planificación temporal implícita:* Esta planificación se basa en decisiones locales, es decir, cada planificador local toma sus propias decisiones de acuerdo con la llegada de eventos.

- *Planificación híbrida:* Planificación que comparte técnicas de las dos anteriormente descritas.

La figura 1.4 ilustra como se representa esquemáticamente la taxonomía de la planificación temporal.

Figura 1.4: Planificación temporal

El segundo grupo en el que hemos dividido la planificación de aplicaciones paralelas es la planificación espacial.
Esta planificación se encarga de decidir en qué nodos se ejecutarán los trabajos y cómo se planificarán. Atendiendo a la figura 1.5, podemos dividir este grupo en 2 subgrupos, la distribución de nodos y la planificación de trabajos.

Figura 1.5: Planificación espacial

Teniendo en cuenta la distribución de nodos, encontramos la necesidad de particionar y seleccionar los nodos para su correcta distribución. La figura 1.6 presenta una solución que solventa este tipo de problemas.

Figura 1.6: Distribución de nodos

El tipo de particionamiento podemos clasificarlo en estático y variable.

- *Particionamiento estático:* Las particiones de los nodos son definidas por el administrador y no cambian en el tiempo a no ser que sea el propio administrador quien las cambie expresamente.

- *Particionamiento variable:* Este tipo de particionamiento se realiza en función de los requerimientos de las aplicaciones a través de las cuales se construyen particiones que se ajustan a esos requerimientos.

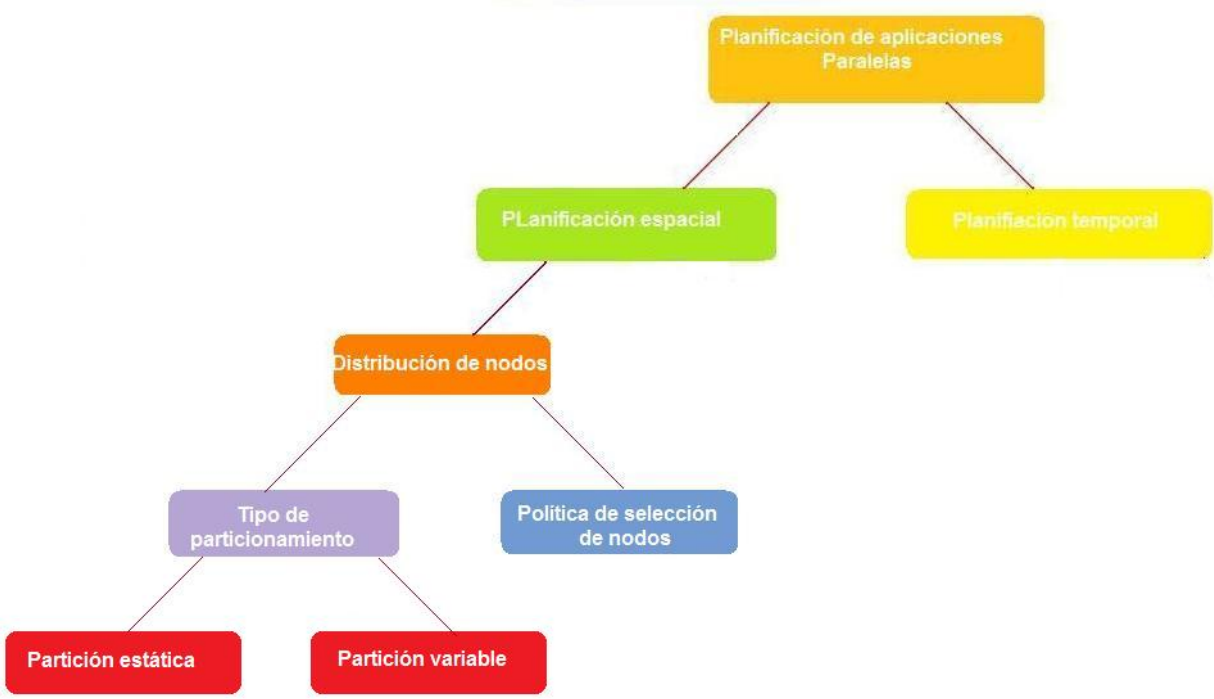


Figura 1.7: Tipo de particionamiento

La política de selección de nodos, figura 1.8, intenta decidir qué nodos concretos del conjunto total de nodos serán los escogidos para lanzar el trabajo. Podemos dividir esta política en 2 grupos, selección binaria y selección discreta.

- *Selección binaria:* Es la política más simple, considera que un nodo o bien está libre o ocupado, y no considera la posibilidad que las aplicaciones puedan compartir nodos.

- *Selección discreta:* Son las políticas donde se considera un grado de multiprogramación (Multi Programing level) mayor a 1. Este tipo de aplicaciones he de combinar el tiempo compartido con el espacio compartido.

La figura 1.8 muestra la taxonomía completa de distribución de nodos conjugando el tipo de particionamiento y la política de selección de nodos

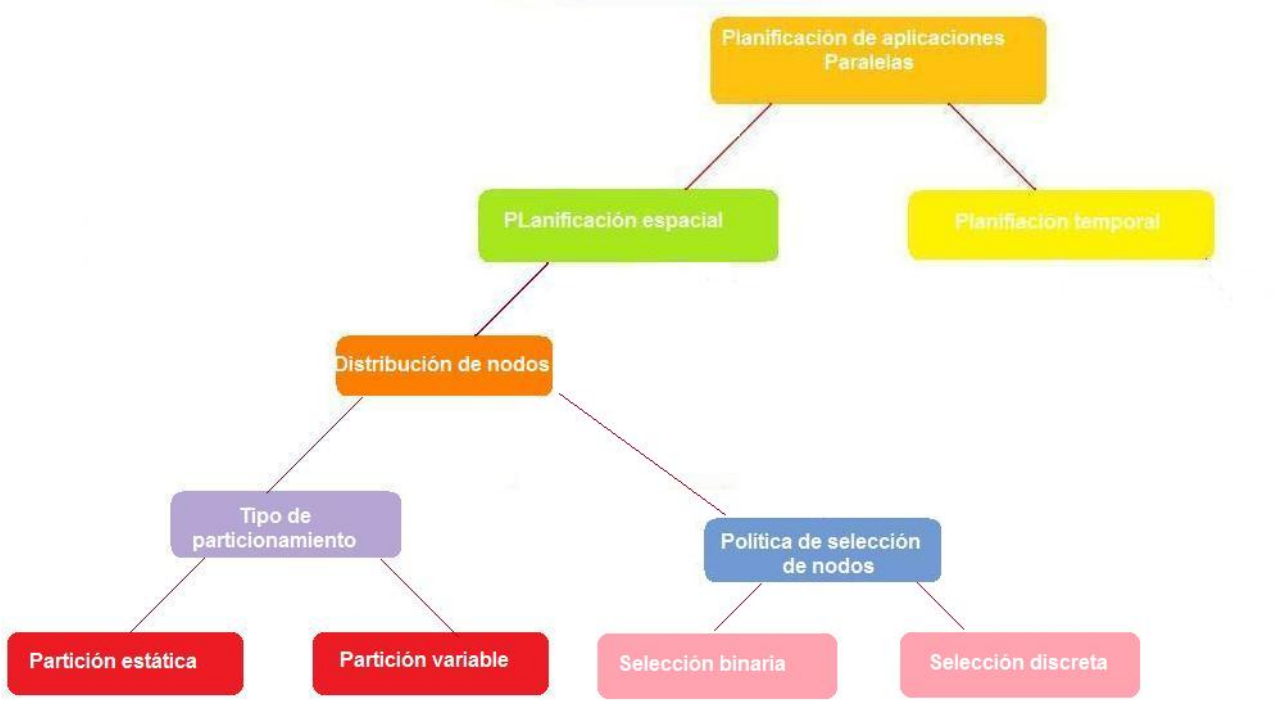


Figura 1.8: Distribución de nodos

Una vez descritas las políticas de selección de nodos, pasamos a desarrollar la funcionalidad y alternativas de las políticas de planificación de trabajos.
Las políticas de planificación de trabajos, son las encargadas de ordenar los trabajos para su ejecución, a la vez que decidir por algún criterio de selección qué trabajo de la cola de ejecución se debe ejecutar en cada momento. Dentro de este grupo, podemos encontrar políticas de orden y políticas de selección.
Las políticas de orden son las encargadas de situar un trabajo en alguna posición relativa a los demás trabajos de la cola de ejecución. Existen infinidad de políticas de orden diferentes, pero en este estudio hemos seleccionado alguna de las importantes, como son la FCFS, SJF y SNPF.

- *FCFS (First Come First Served):* Los trabajos son ejecutados en el orden que llegan al sistema.

- *SJF (Shortest Just First):* Los trabajos se ordenan de forma creciente en función del tiempo de ejecución estimado.

- *SNPF (Smallest Number of Processes First):* Los trabajos se ordenan en función de la cantidad de procesadores que necesitan.

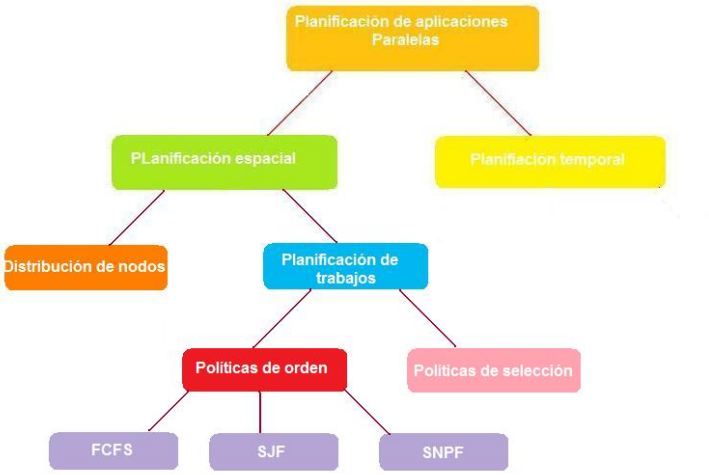


Figura 1.9: Políticas de orden de planificación de trabajos

Una vez ordenada la cola de espera necesitamos seleccionar los trabajos que se encuentran en ella para su ejecución.
Aunque elegir el primero pueda parecer la mejor opción, no siempre es cierto, ya que dependiendo de los recursos que necesite un trabajo para ejecutarse puede frenar a los demás. Para poder seleccionar la mejor opción, las políticas de selección de trabajos necesitan conocer el estado actual del cluster. Algunas políticas van más allá y consideran una mayor cantidad de información. Normalmente este tipo de sistemas más complejos consideran el comportamiento futuro del sistema, basado en algún tipo de estimación, a efectos de establecer los criterios de asignación y clasificar las políticas de selección. Estas políticas se dividen en políticas de estado actual y políticas de estado futuro.
Dentro de las políticas de estado actual, encontramos la First Fit y la Best Fit.

- *First Fit:* Se recorre la cola hasta encontrar el primer trabajo que quepa en el espacio de procesadores.

- *Best Fit:* Se recorre la cola de principio a fin para buscar el trabajo cuyos requerimientos se aproximen más al conjunto de recursos libres actualmente en los nodos.

Las políticas de estado futuro se basan en intentar mejorar el rendimiento del sistema, necesitando predecir el estado futuro del cluster. Dos alternativas a este tipo de políticas son la QoS y la Backfilling.

- *QoS (Quality of service):* Busca garantizar un conjunto de recursos durante un tiempo limitado como parte de los requerimientos de una aplicación para poder ejecutarse.

- *Backfilling:* Es una de las políticas más utilizadas. Se basa en el concepto de adelantar trabajos; es decir, si un trabajo puede ejecutarse antes que otros que se encuentran en la cola de espera por delante, se ejecutará siempre y cuando no retrase el tiempo de inicio de dichos trabajos. Dentro de las políticas de Backfilling podemos encontrar varias variantes del algoritmo básico:

  - *Backfilling EASY:* Un trabajo siempre puede adelantarse en la cola de trabajos si eso no retrasa a los trabajos que se encuentren en la cabeza de la cola de ejecución.

  - *Backfilling Conservativo:* Un trabajo sólo puede adelantarse en la cola si no retrasa a ninguno de los trabajos que se encuentran por delante de el.

  - *Backfilling de múltiples colas:* Inclusión de múltiples colas en lugar de una única. De esta forma se clasifican los trabajos en función de su tamaño.

  - *Backfilling expropiativo:* Un trabajo puede perder los recursos asignados en función de la prioridad de los trabajos que van llegando a la cola de ejecución.

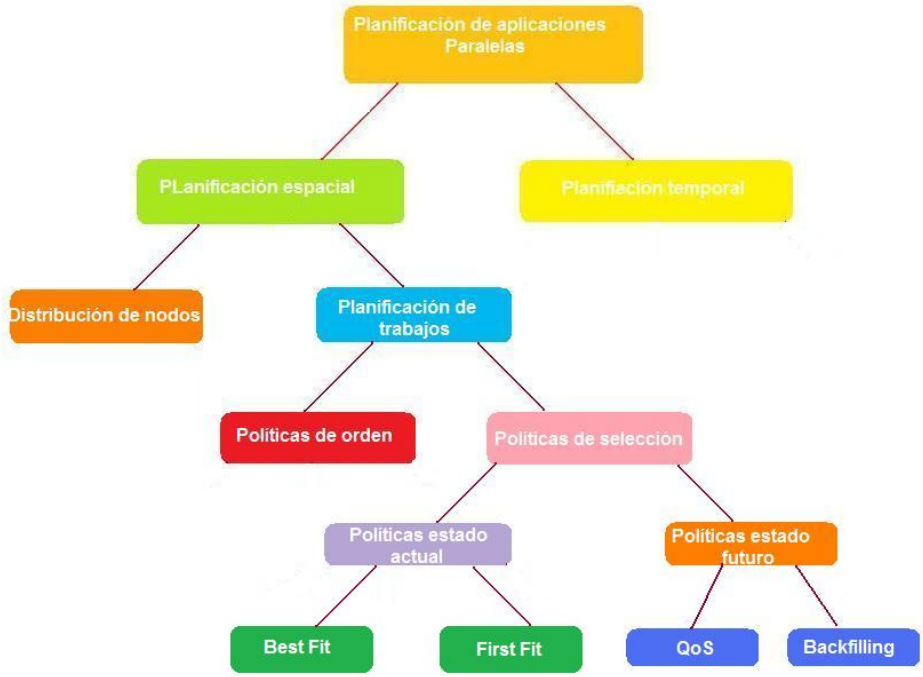


Figura 1.10: Políticas de selección

Toda esta variedad en cuanto a las posibilidades de ajuste de los entornos de planificación, definirán opciones a seleccionar por el usuario o administrador del cluster a efectos de sintonizar el sheduler a su carga de trabajo, o aplicación en particular.

1.2 Objetivos del trabajo

La metodología de trabajo adoptada por el usuario de aplicaciones paralelas, y su interrelación con el subsistema de cómputo mediante la utilización del paradigma de entornos gráficos ha sido uno de los objetivos de este estudio. Desde el comienzo del estudio, se aposto por la utilización de entornos gráficos, ya que facilita la comunicación con el cluster permitiendo un manejo del mismo mucho más versátil.

Actualmente, la comunicación con clusters puede ser efectuada de dos maneras distintas, una de ellas se realiza vía consola remota a través de ssh, a algún punto de logín que conecta con el sistema de colas del scheduler, mientras que la otra se efectua a través de entornos gráficos (GUI).
La forma tradicional de comunicación con clusters hasta la aparición de entornos GUI, era mediante consola remota a través de ssh. En la figura 1.11 podemos observar el esquema de comunicación. Dicha forma de trabajo era tediosa y suponía un trabajo previo considerable como tener que escribir los scripts de especificaciones para cada aplicación que se enviaba al cluster para su posterior ejecución. No sólo presentaba esa desventaja, sino que para consultar las estadísticas, era necesario enviar el comando mediante terminal y la respuesta nos era enviada al terminal en modo de texto, lo cual dificultaba su lectura.

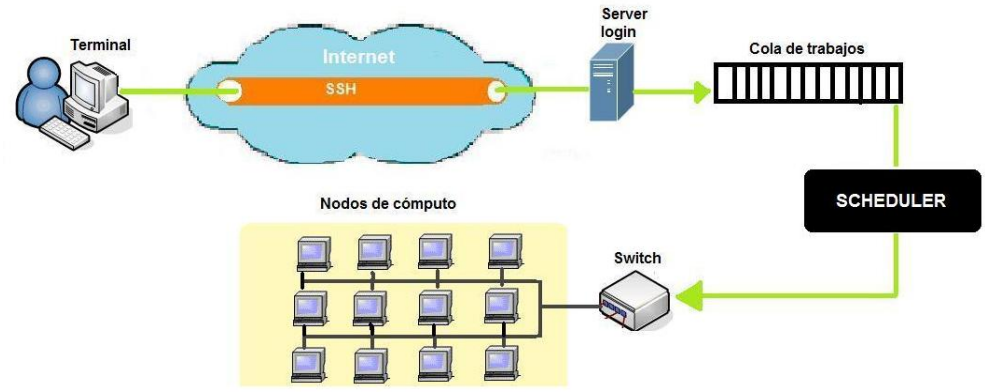


Figura 1.11: Esquema de comunicación mediante consola

Con la llegada de las interficies gráficas, el proceso de comunicación anterior se vio claramente simplificado, ya no es necesario crear un script de especificaciones, sino que dichas especificaciones se pueden seleccionar a través de las diferentes ventanas del entorno. Los entornos gráficos no sólo muestran esa ventaja, sino que para la consulta de estadísticas al cluster resulta mucho más cómodo, ya que son mostradas mediante gráficos, lo que supone una lectura más clara e intuitiva. Otra de las ventajas que nos ofrecen los sistemas gráficos, es que no sólo se pueden utilizar en modo usuario para enviar trabajos y consultar estadísticas, sino que permiten entrar en modo administrador y configurar los parámetros del cluster, este sistema nos evita tener que editar los archivos de configuración a mano. También permiten, la parametrización automática de aplicaciones, el uso de simuladores y la selección de políticas.

Gracias al amplio abanico de prestaciones que nos ofrecen las interficies gráficas, hace que resulte bastante atractiva su utilización para el manejo de clusters.

En cuanto al proceso de autentificación, el servidor de login desaparece para dejar paso a un servidor web, dicho servidor posee una aplicación totalmente transparente para el usuario donde lo unico que tiene que conocer el usuario es la dirección y el puerto donde estará escuchando la aplicación del servidor. La figura 1.12 describe como es el proceso de conexión.

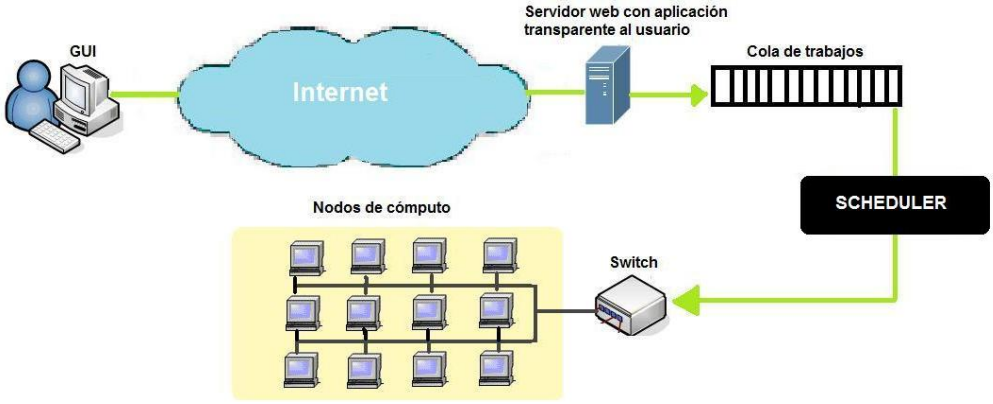
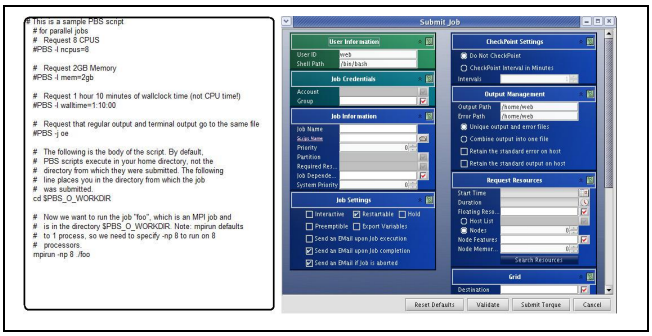
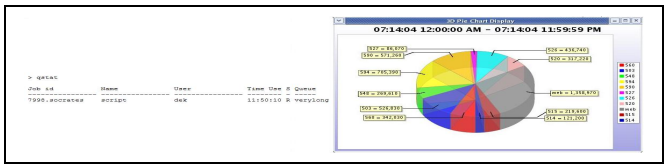


Figura 1.12: Esquema de comunicación utilizando interficies gráficas

Las figuras 1.13 y 1.14 ilustran una comparación de los dos entornos, las figuras 1.13.a y 1.14.a muestran como se realizarían la comunicación con el cluster en modo de texto, mientras que las figuras 1.13.b muestran su equivalente en modo gráfico.



Figuras 1.13.a y 1.13.b: Formas de envio de trabajos al cluster



Figuras 1.14.a y 1.14.b: Presentación de estadísticas

Independientemente de la utilización de un entorno u otro para la comunicación con el cluster, se hace necesario conocer los gestores de cola que estamos utilizando, pues aunque no tengamos que construir los scripts de especificaciones, es necesario parametrizar los trabajos que enviamos al cluster para su correcta ejecución. El cluster, debe de recibir la misma información de parametrización de cada trabajo independientemente del entorno de comunicación utilizado. Cada trabajo o aplicación tendrá una parametrización específica basándose en las características del tipo de aplicación que deseamos ejecutar.

A la hora de seleccionar la política de planificación, puede resultar muy complicado escoger la política idónea, dependiendo del tipo de aplicaciones que tengamos en la cola de ejecución, para no frenar la ejecución de otros trabajos más prioritarios. Para intentar solventar este problema se puede recurrir a la simulación. La simulación, mediante un archivo de trazas de las aplicaciones que queramos ejecutar en el cluster y otro archivo de recursos, donde estarán los recursos físicos del clusters, nos permite tener una copia exacta de nuestro cluster real, donde poder experimentar con diferentes políticas de planificación, hasta encontrar la más idónea para nuestras aplicaciones.

Una vez lanzada la simulación, podemos ir observando el estado de los trabajos en la cola de ejecución y como van evolucionando en el tiempo.
La simulación no sólo resulta beneficiosa para escoger la política de planificación, sino que es de gran interés para observar los recursos que consume nuestra aplicación y poder ajustar la parametrización para lanzar las aplicaciones en el cluster físico, ajustándola lo mejor posible a los recursos que necesite, ya que sobredimensionar recursos supone un gasto económico importante.

Como objetivo secundario se ha querido dar una visión global de los componentes de un cluster, interrelacionandolos entre si y viendo el objetivo de cada uno de esos componentes de forma general.

Debido a la cantidad de alternativas a seleccionar de los entornos de desarrollo, que tienen que ver con: la parametrización de aplicaciones, caracterización de la carga, especificación de la arquitectura de los nodos a utilizar, selección de las políticas de planificación, elección de las métricas adecuadas de perfomance (usuario, sistema), manejo de simuladores y herramientas de monitorización de resultados; añadiendo la falta de información de las herramientas de instalación y de uso de entornos de forma adecuada(parametrización de la aplicación para el simulador); así como la no existencia de ejemplos de uso del sistema extremo a extremo; creemos que este trabajo puede servir de referencia inicial a usuarios que deseen utilizar este tipo de entornos de desarrollo de aplicaciones paralelas.

1.3 Modelo de desarrollo

El desarrollo de este proyecto está planteado por etapas lineales, ya que hemos seguido la estrategia de cascada de desarrollo de proyectos.
Debido a la interrelación que existe en alguna de las etapas se ha decidido trabajarlas en paralelo para obtener mejores resultados.
En la siguiente tabla se presentan las diferentes etapas seguidas para la realización del proyecto y en un apartado posterior presentaremos la sucesión en las etapas en el tiempo.

Estudio del proyecto
Estudio de la viabilidad
Requerimientos
Estudio de alternativas
Desarrollo y codificación
Estudio de los resultados
Conclusiones de los resultados
Memoria

Tabla 1.1: Estapas de desarrollo

1.4 Distribución temporal de tareas (Gantt)

En este apartado se describirá en que consiste cada etapa del proyecto y se mostrará el tiempo empleado en realizar cada tarea.

- *Estudio del proyecto:* En esta etapa se ha estudiado como íbamos a enfocar el proyecto y que se iba a realizar exactamente, ya que el campo de la computación paralela es muy extenso.

- *Estudio de la viabilidad*: Una vez decidido el tema concreto a investigar, en esta etapa se ha realizado un riguroso estudio para ver si era posible llevar a cabo el proyecto en el tiempo estimado y era asequible de efectuar para una sola persona.

- *Requerimientos:* En esta etapa se ha elaborado una lista de todos los requerimientos que eran necesarios para la realización del proyecto y así poder asegurarnos que disponíamos de todo lo necesario para poder realizarlo con satisfactoriamente.

- *Estudio de alternativas:* En esta etapa se ha decidido que componentes software se iba a utilizar para el desarrollo del proyecto. Se estudiaron las diferentes alternativas y unas vistos las ventajas e inconvenientes de todos ellos se seleccionó el que más nos interesaba atendiendo a nuestras necesidades.

- *Desarrollo y codificación:* En está etapa se llevó a cabo parte práctica del proyecto. Configuramos un cluster atendiendo al software elegido previamente. Una vez configurado el cluster procedimos a la parametrización de las aplicaciones y su ejecución en el cluster.

- *Estudios de los resultados:* Una vez obtenidos los resultados en esta etapa nos dedicamos a estudiar los resultados obtenidos resultados y compararlos.

- Conclusiones de los resultados: Esta etapa la dedicamos a extraer las conclusiones una vez entendidos y estudiados los resultados de la etapa anterior.

- Memoria: La hemos ido elaborando a lo largo de todo el proyecto.

En la siguiente figura se muestra el tiempo empleado en cada una de las etapas del proyecto.

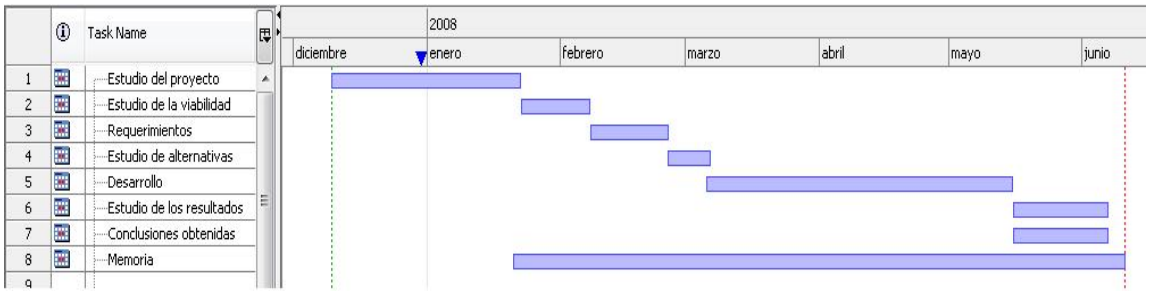


Figura 1.15: Diagrama de gannt

1.5 Estructura de la memoria

En esta memoria se pretende explicar los pasos seguidos desde el inicio del proyecto hasta su finalación.

- Primer capítulo: Hemos pretendido dar una breve descripción del problema a tratar. Se exponen las tareas realizadas en el proyecto así como su planificación temporal.

- Segundo capítulo: Lo dedicaremos a exponer todos los factores que influyen en el desarrollo de la ejecución de las aplicaciones en el cluster.

- Tercer capítulo: Se centra en el estudio del entorno de desarrollo, y la metodología de trabajo utilizada en ese tipo de entornos.

- Cuarto capítulo: Se centra en explicar la metodología llevada a cabo a lo largo del estudio de forma práctica.

- Quinto capítulo: Se exponen las conclusiones obtenidas en el estudio y las líneas abiertas a las que este trabajo ha dado lugar.

Capítulo 2

Componentes básicos de un sistema de desarrollo para clusters

2.1 Introducción

Como definición del termino "cluster", hemos escogido la del Dr. Rajkumar Buyya, por ser una definición formal, pero sin entrar en detalles muy técnicos del concepto cluster.
Buyya, en uno de sus libros titulado *High Performance Cluster Computing Volume 1*, define un cluster como: "un tipo de sistema de procesamiento, ya sea paralelo o distribuido, que consta de un conjunto de ordenadores, conectados mediante una infraestructura de redes de alta velocidad que funcionan como una sola computadora".

Como último punto de esta introducción, la figura 2.1 muestra el esquema de un cluster de altas prestaciones, con todos los componentes esenciales para el correcto funcionamiento del mismo, de forma que se observe con claridad la interrelación entre ellos, lo que supondrá que a la hora de explicar dicho componente el lector pueda situarlo adecuadamente y vea sus dependencias con otros componentes.

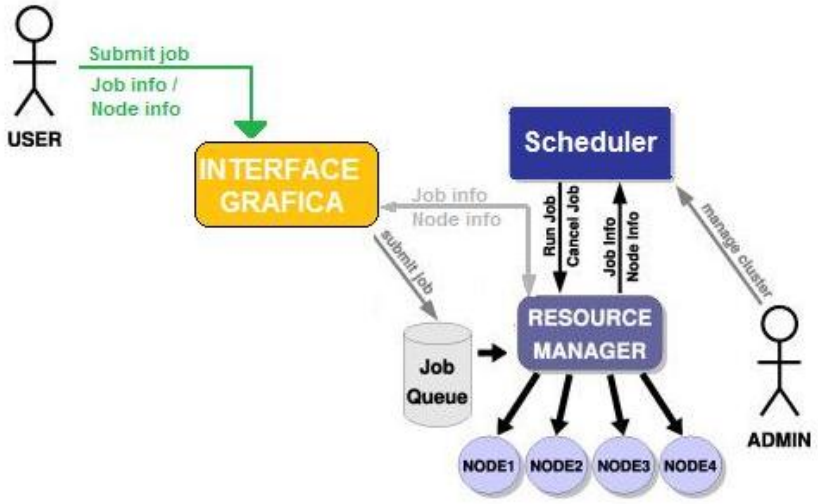


Figura 2.1: Esquema de un cluster

2.2 Gestores de colas

Un gestor de colas es un sistema de administración y distribución de los trabajos que se envían al cluster. En este apartado se exponen los gestores de colas más extendidos y se presentan sus principales ventajas e inconvenientes.

2.2.1 PBS

Gestor de colas originalmente diseñado por la NASA que ha llegado a ser un estándar *de facto* en clusters Linux.
Proporciona una serie de herramientas para la gestión de trabajos batch, utilizando una unidad de programación de tareas. Además, permite el enrutamiento de estos trabajos a través de diferentes computadores.
Cuenta con capacidades para definir e implementar políticas sobre la utilización de los recursos disponibles. El 90% de los clusters de producción Linux utilizan este sistema de colas por ser el más extendido y fiable.
PBS está compuesto básicamente por dos componentes, los comandos de usuario y los demonios del sistema.
La figura 2.1 ilustra la interrelación entre todos los componentes de PBS.

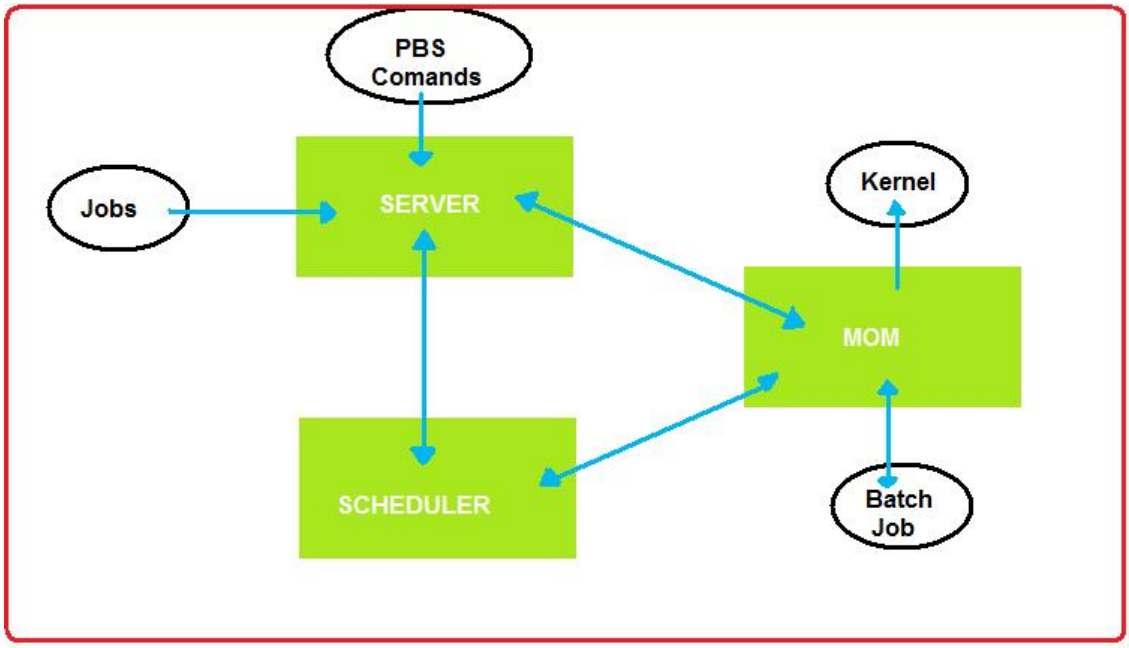


Figura 2.1: Comunicaciones entre los componentes de PBS

Los cuadrados marcados en verde representan los demonios de PBS. Como podemos apreciar en la figura 2.1, PBS se compone de tres demonios, dos de ellos son necesarios para su correcto funcionamiento mientras que el scheduler puede ser reemplazado por un planificador externo.

- *SERVER:* Demonio encargado de recibir los trabajos a ejecutar y estar a la espera de comandos de usuario.

- *MOM:* Demonio que se ejecuta en cada nodo de cómputo y que envía y recibe los trabajos a ejecutar

- *SCHEDULER:* Planificador que decide qué trabajos se van a ejecutar en el cluster dependiendo de la política seleccionada.

2.2.2 SLURM

Gestor open-source diseñado para clusters GNU/Linux de todos los tamaños. Provee algunas funcionales clave que hacen muy interesante su uso:

- Ubica acceso exclusivo y/o no exclusivo de los recursos (nodos de cómputo) a los usuarios por alguna duración de tiempo de manera de que ellos puedan realizar algún trabajo.

- Provee un ambiente de trabajo para comenzar, ejecutar y monitorear trabajos (típicamente paralelos) en un conjunto de nodos específicos.

- Se encarga de arbitrar peticiones a recursos conflictivos, mediante el manejo de una cola de trabajos pendientes.

- Provee un API para la integración con planificadores externos tales como el MAUI Scheduler. Mientras en otros gestores de cola no existe.

- Es lo suficientemente simple para el usuario final, entender su código fuente y agregar funcionalidades.

- Permite manejar aspectos clave de las arquitecturas multicore como puede ser la afinidad.

SLURM está compuesto básicamente por dos tipos de demonios y 5 comandos de usuario. En la siguiente figura podemos ver los diferentes componentes de SLURM y la interrelación entre cada uno de ellos.

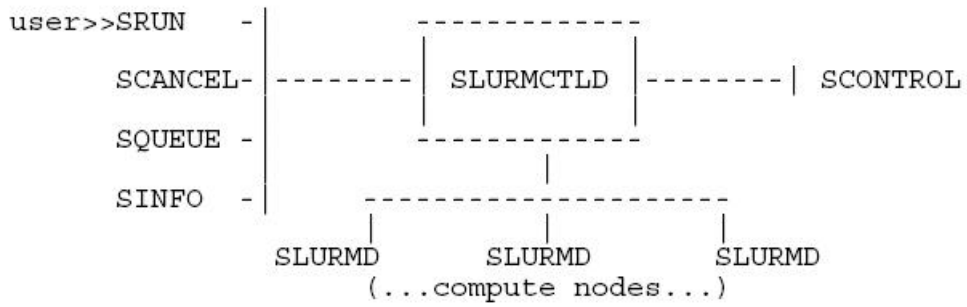


Figura 2.2: Comunicaciones entre los componentes de PBS

Como podemos apreciar en la figura anterior, SLURM se compone de dos demonios, SLURMCTLD y SLURMD.

- *SLURMCTLD:* Demonio encargado del control principal de SLURM. A diferencia del demonio servidor de PBS, SLURMCTLD tiene la propiedad de ser Multi-Threaded.

- *SLURMD:* Demonio que se ejecuta en cada nodo de cómputo y que envía y recibe los trabajos a ejecutar.

2.2.3 LoadLeveler

Gestor de colas diseñado por IBM, que se caracteriza por su facilidad para construir, subministrar y procesar los trabajos en el cluster.
Permite la ejecución tanto de trabajos serie como paralelos (MPI) y es fácilmente escalable a miles de procesadores.
Loadleveler fue uno de los primeros sistemas en incorporar el algoritmo de planificación backfill que ha continuado mejorando la escalabilidad, velocidad y rendimiento de dicho algoritmo para su gestor de colas.
Fue el primer sistema en ofrecer una completa API de scheduling que ofrecía una amplia flexibilidad para la programación del scheduler. Actualmente ofrece un completo conjunto de API's que hace que se sea una atractiva posibilidad a tener en cuenta a la hora de escoger un gestor de colas.
Posee una mejor gestión de tolerancia a fallos y auto-reparación, mediante el uso de checkpoints.
Proporciona una interficie gráfica de usuario, que a pesar de ser muy rudimentaria y ejecutarse en la consola, fue muy atractiva cuando se implementó, ya que permitía subministrar trabajos y hacer tareas de monitorización fácilmente.
Como desventajas de este gestor colas, hay que hacer notar que los comandos son bastante diferentes a los utilizados por PBS, Torque o Slurm en cuanto a su sintaxis.
Cabe decir también que es software privado y para poder utilizar todo el pack es necesario comprar el producto.

2.2.4 Torque

Basado en PBS y creado por la empresa Cluster Resources Inc. Se caracteriza porque se puede integrar con un atractivo entorno gráfico para enviar trabajos, consultar estadísticas; desplazando para ello la forma de enviar trabajos via SSH por consola a un segundo plano.
Sus principales características son las siguientes:

- Tolerancia a fallos
  - Posee chequeos y manejos de condiciones de fallo.
  - Permite la reparación de muchos errores.

- Tiene un script de soporte para chequear el estado o salud de los nodos.

- Interfaz de planificación
  - Permite integrarse con el scheduler MOAB, lo que le da una inmensa potencia y flexibilidad.
- Escalabilidad

  - Servidor significativamente mejorado para el modelo de comunicación MOM

  - Habilidad para manipular grandes clusters (sobre 15 TF/2,500 procesadores)

Una vez más, su parte negativa es la de ser un programa privado; para conseguir poder obtener toda su potencia, es necesario adquirir la versión comercial.

2.3 Planificadores externos

Podemos definir planificadores externos, como el sistema que se encarga de distribuir los trabajos en el cluster dependiendo de los parámetros de configuración, prioridades y la política de este.
Es el encargado de organizar los trabajos en las colas y lanzarlos al cluster o incluso, si lo considera necesario, interrumpir un trabajo y devolverlo a la cola para lanzar a otro más prioritario en su lugar.
Debido a la complejidad que presenta este componente y al gran número de parámetros de configuración que posee, como puede ser: la selección de la política de planificación, configuración de los parámetros del simulador, configuración del checkpoint, selección del modo de trabajo a utilizar(normal, monitor, simulador), reserva de nodos, número máximo de recursos a utilizar por un determinado tipo de trabajo, etc; hace que resulte uno de los componentes más complejos del cluster y que más dedicación ha tenido en este estudio. En el anexo 6 podemos encontrar la lista completa de parámetros de configuración junto a una breve descripción.

2.3.2 MAUI

MAUI es un planificador externo integrado en el sistema, que periódicamente itera sobre los trabajos pendientes en el gestor de colas. Cada iteración (cuyo intervalo de tiempo varía en función del parámetro RMPPOLL), consulta al gestor de colas el estado de los trabajos en el sistema. A partir de esa información, MAUI ordena los trabajos pendientes de ejecución en función de las políticas de prioridad del sistema, y los ejecuta en el orden de mayor a menor prioridad.
Presenta la ventaja de ser de código abierto, lo que ha facilitado su rápido crecimiento, y actualmente es uno de los planificadores más extendidos. Permite ser integrado en la gran mayoría de gestores de colas, tales como PBS, LoadLeveler, SGE y BPROC.

Es totalmente configurable a las necesidades del usuario. Como características destacaremos las siguientes:

- Permite definir diferentes políticas con diferentes prioridades.
- Posee una interface Metascheduling.
- Permite configurar múltiples políticas backfill.
- Diferentes modos de uso como son el normal, test o monitor.
- Posee un simulador de scheduler que permite analizar workloads para diferentes políticas.

La figura 2.3 ilustra la diferencia de usar un planificador externo como MAUI con otro incorporado en el gestor de colas.

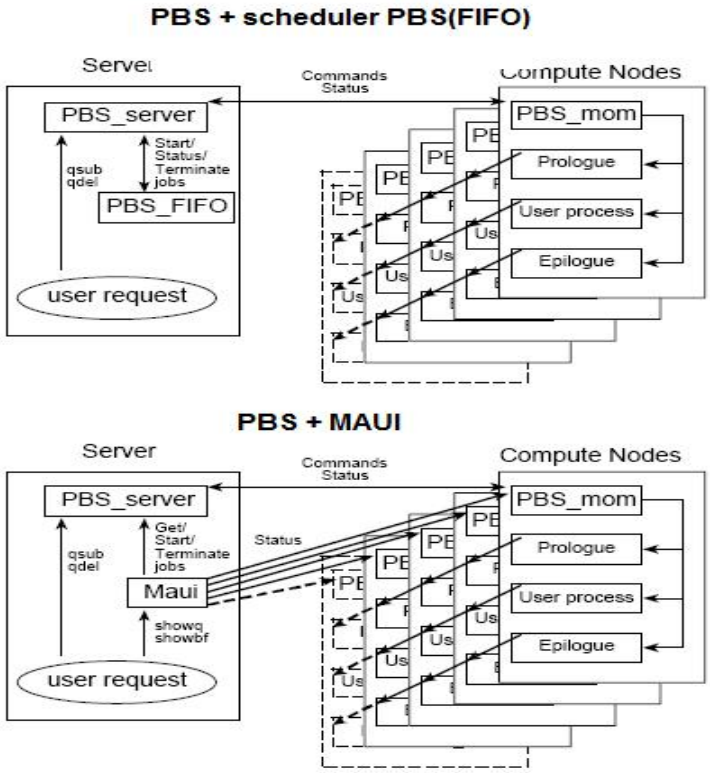


Figura 2.3: Diferencia entre usar MAUI y un planificador incorporado

2.3.3 MOAB

Planificador de la empresa Cluster Resource Inc. Es la versión comercial de MAUI, su antecesor.
Se ha impuesto rápidamente gracias a su gran potencia y versatilidad de configuración.
Al igual que MAUI, MOAB incorpora un simulador, el cual permite poder probar en un tiempo razonable un extenso abanico de configuraciones posibles del cluster.

Puesto que las características son básicamente las mismas que en MAUI no las volveremos a comentar, sólo remarcaremos algunas mejoras y optimizaciones que se han incorporado:

- Maximiza las prestaciones del cluster gracias a la alocatación inteligente de recursos.
- Permite reservas recursos por un periodo de tiempo determinando para algún usuario en concreto.
- Permite acciones arbitrarias asociadas a los eventos del cluster. Estas acciones se conocen como trigers.

2.3.4 LoadLeveler Scheduling

Planificador que viene incorporado en el paquete comercial de LoadLeveler.
Es el menos avanzado de los mostrados en esta sección, pero su facilidad de configuración y su simplicidad hace que sea atractivo cuando no se requieren actividades demasiado complejas del cluster.

2.4 Librerías de paso de mensajes

Permiten la comunicación entre los diferentes nodos de cómputo. Es necesario tener instaladas las librerías en todos los nodos de cómputo para su correcto funcionamiento. Seguidamente se presentan los 2 estándares más utilizados.

2.4.1 Standard MPI

Librería de paso de mensajes propuesta como estándar por un comité de vendedores, implementadotes y usuarios. Implementa las siguientes características y funcionalidades:

- Colección de funciones que ocultan detalles de bajo nivel, tanto software como hardware.
- Diseñado para obtener un alto rendimiento tanto en máquinas paralelas como en clusters.
- La unidad básica de paralelismo son los procesos independientes.
- Tienen espacios de memoria independientes.
- Intercambio de datos y sincronización mediante paso de mensajes.
- A cada proceso se le asigna un identificador interno propio.

- Proporciona una funcionalidad flexible, incluyendo diferentes formas de comunicación, rutinas especiales para comunicaciones "colectivas" y la habilidad de usar tipos de datos y topologías definidas por el usuario dentro de las comunicaciones.

- Programación en Fortran y C. En la revisión del estándar (MPI-2) se soporta C++ y Fortran 90.

La figura 2.4 ilustra el proceso que realiza MPI para enviar y recibir mensajes de un proceso a otro utilizando la librería de paso de mensajes MPI. Lo interesante de la figura, es observar como cada mensaje es guardado en un buffer independiente antes de ser enviado al proceso receptor, lo que hace que cada proceso tenga un espacio de memoria independiente.

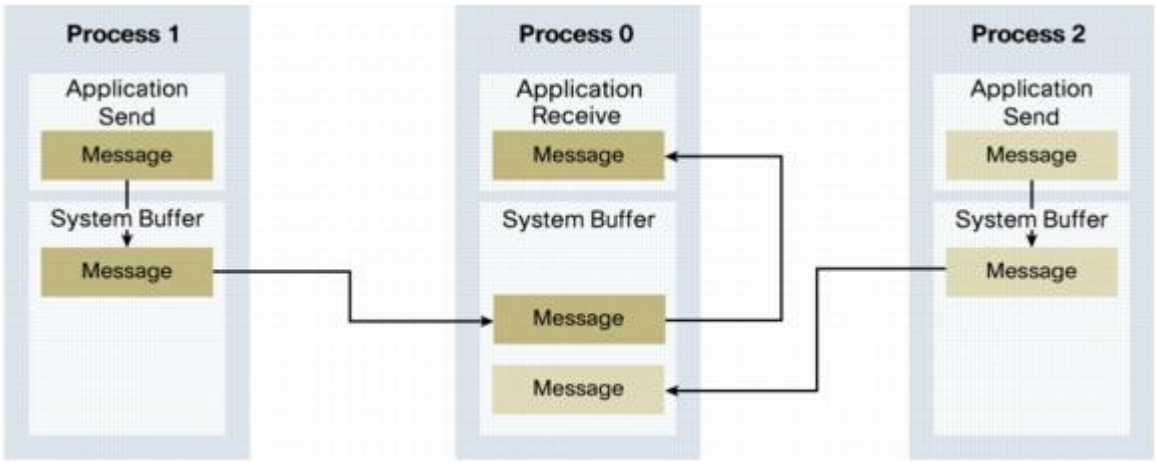


Figura 2.4: Paso de mensajes utilizando MPI

2.4.3 PVM (Parallel Virtual Machine)

Software de libre distribución creado por Oak Ridgre en 1989 que permite ejecutar aplicaciones paralelas sobre máquinas distribuidas y heterogenias. Sus principales características son los siguientes:

- Existe un conjunto de nodos (hosts) susceptibles de poder ser usados en tareas de computación, definido por el usuario.

- El usuario selecciona el conjunto de máquinas donde se ejecutarán la tarea de computación paralela. Dicho conjunto puede alterarse en tiempo de ejecución, añadiendo o quitando máquinas, esto es importante para la tolerancia a fallos.

- Acceso semitransparente al hardware: Podemos aprovechar las características de cada nodo para ejecutar determinados cálculos.

- Computación basada en el proceso.

- Modelo de paso de mensajes explícito.
- Soporte para arquitecturas heterogéneas, en términos de máquinas, redes y aplicaciones.
- Soporta los lenguajes C, C++ y Fortran.

La siguiente figura 2.5 muestra el proceso de comunicación entre diferentes nodos distribuidos y conectados mediante una red.

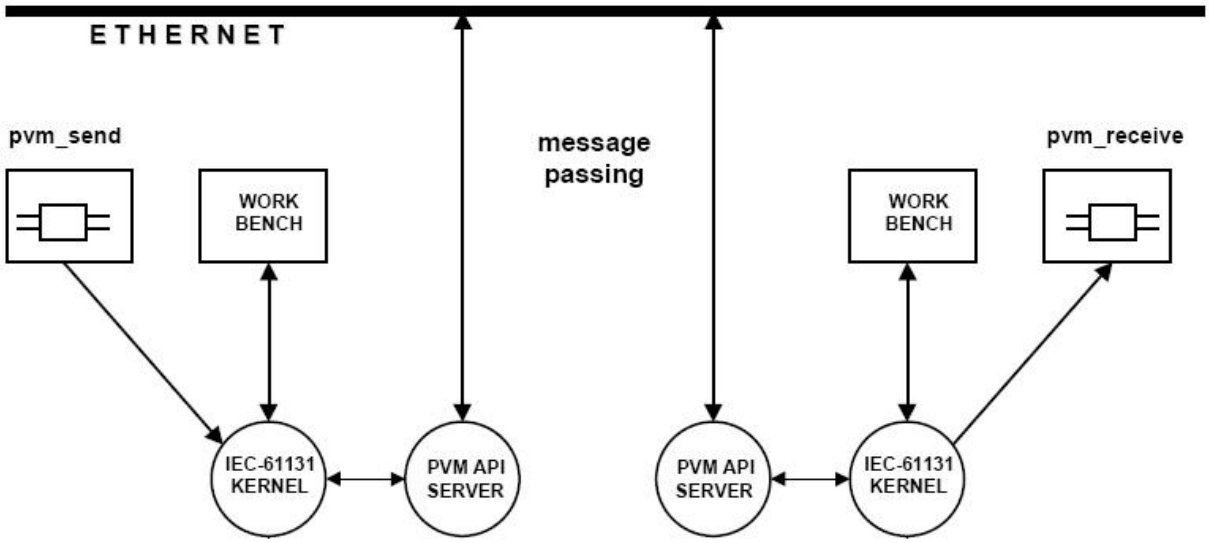


Figura 2.5: Paso de mensajes utilizando PVM

2.5 Sistema operativo

En la elección del sistema operativo, tenemos que tener en cuenta que este sea multiproceso y multiusuario. Actualmente podemos escoger entre Microsoft Windows (distribuciones Server edition) o sistemas operativos de código abierto como GNU/Linux. Actualmente el 80% de los clusters están montados sobre plataformas GNU/Linux. Esto no sólo es debido al menor coste económico que presenta utilizar sistemas de código abierto frente a software privativo, sino que también tiene un gran peso en la elección la facilidad para configurar el sistema operativo a nuestras necesidades, ya que GNU/Linux nos brinda la posibilidad de poder configurar o cambiar el sistema operativo a nuestras necesidades.
En este proyecto se han presentado y utilizado gestores de colas basados en GNU/Linux (aunque actualmente existen versiones para Windows) por ser los más extendidos y utilizados, pero también es posible encontrar gestores de colas exclusivos para sistemas operativos Microsoft Windows, como es el caso de Windows Computer Cluster Server (CSS). Windows CSS es una plataforma basada en estándares que funciona bajo Windows 2003 Server y está siendo utilizada para el cómputo paralelo.

La elección de la versión en sistemas Windows no tiene discusión al respecto, ya que estamos obligados a utilizar una de la dos versiones que disponen de la versión servidor del operativo (Windows 2000 server edition o Windows 2003 server edition).
En cuanto a la elección de sistemas GNU/Linux, tenemos un amplio abanico de posibilidades, siendo Debian y Fedora las distribuciones más utilizadas debido a ser dos de las versiones más estables y potentes, aunque actualmente se está imponiendo con fuerza Ubuntu Server para la instalación de clusters.

2.6 Redes de interconexión

La conexión entre los diferentes nodos del cluster es un punto importante a tener en cuenta, ya que dependiendo del tipo de tecnología que utilicemos y de las aplicaciones que queramos ejecutar, puede variar sustancialmente el resultado obtenido.
Existen diferentes tecnologías para realizar la conexión entre los diferentes nodos; las más utilizadas son las siguientes:

- *Fast Ethernet:* LAN introducida en 1994, como la evolución de ethernet. Proporciona un ancho de banda de 10 Mbits/segundo. Mediante el uso de conmutadores y hubs se pueden definir varios dominios de colisión separados.

- *Gigabit Ethernet:* Aparecida en 1998, proporciona un ancho de banda de 1 Gbit/segundo. Esta tecnología está basada en conmutadores punto a punto.

- *Myrinet:* Red diseñada por la empresa Myricon. Proporciona latencias de 3 microsegundos y un ancho de banda que puede oscilar entre los 2Gbps y los 10Gbps. Una de sus principales características, además de su rendimiento, es que el procesamiento de las comunicaciones de red se hace a través de chips integrados en la tarjetas Myrinet, descargando a la CPU parte del proceso de comunicaciones. Las especiales características de Myrinet hacen que sea altamente escalable, gracias a la tecnología existente de conmutadores y routers. La inmensa mayoría de los clusters del top500 poseen una red de interconexión de este tipo.

- *Infiniband:* Estándar que define un nuevo sistema de interconexión a alta velocidad punto a punto basado en switches. Proporciona un ancho de banda entre 2.5 Gbits/segundo y 30 GBits/segundo. Rompe con el modelo de E/S basado en transacciones locales a través de buses y apuesta por un modelo basado en paso de mensajes a través de canales.

2.7 Simuladores

2.7.1 Simulador de MOAB

Permite simular el comportamiento de un cluster sin necesidad de tener como soporte un gestor de recursos.
El simulador, para poder trabajar se basa en dos ficheros de configuración, el fichero de recursos y el fichero de carga.

El fichero de recursos (Resources Trace File), describe la configuración de los recursos físicos del cluster, como son el estado del nodo, el número de procesadores, el tamaño de memoria física, el tamaño de memoria virtual, la arquitectura del nodo, etc.
Mientras que el fichero de carga (Workload trace) describe todos los aspectos de la carga que queremos simular junto con los recursos necesarios para ser ejecutada.

Una vez tenemos parametrizados esos dos ficheros, el simulador se pone a trabajar. Es él mismo el encargado de ejecutar el trabajo y simularlo, es decir, el usuario no puede simular ningún trabajo que no este parametrizado; y lo único que puede hacer el usuario es consultar al simulador para obtener datos de él, pero en ningún momento puede subministrarle nuevos trabajos vía comando qsub o msub.

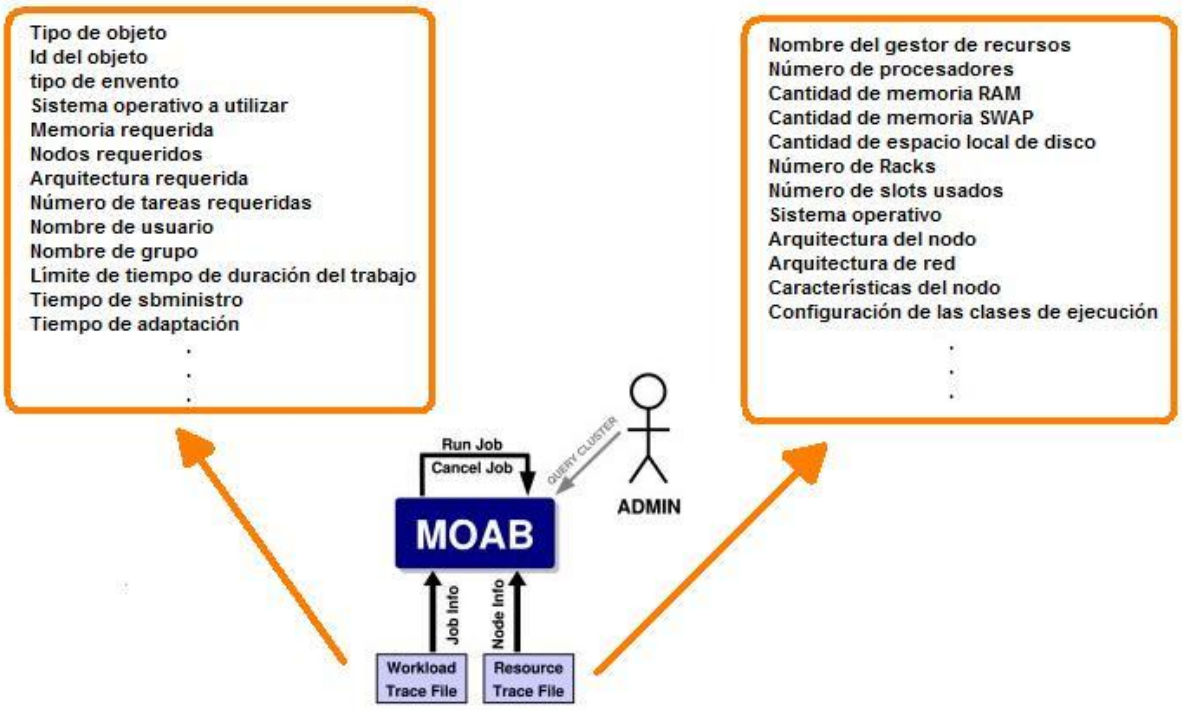


Figura 2.6: Componentes del simulador

Como podemos apreciar en la figura 2.6, debido al gran número de parámetros que hay que subministrar a los archivos de trazas y su elevada complejidad, añadido al elevado número de parámetros de simulación que presenta el scheduler, hace que el proceso de simulación no sea trivial, ya que es necesario conocer todas las opciones de simulación del scheduler para poder simular correctamente las aplicaciones. La tabla 2.1 muestra los parámetros más importantes de simulación que posee el scheduler, para una referencia completa se puede consultar el anexo 6 de este trabajo.

Parámetro	Descripción
SIMULATIONSHUTDOWN	La simulación se termina cuando la cola queda vacía
SIMCPUSCALINGPERCENT	Especifica si incrementar o decrementar el reloj del runtime de cada trabajo

SIMIGNOREJOBFLAGS	Especifica que atributos o criterios ignorará el simulador del fichero de trazas
SIMINITIALQUEUEDEPTH	Especifica cuantos trabajos pueden estar en la cola de ejecución con estado IDLE durante la simulación
SIMJOBSUBMISSIONPOLICY	Especifica como el simulador subministra nuevos trabajos a la cola durante la simulación
SIMNODECOUNT	Especifica el máximo número de nodos que pueden utilizarse para la simulación
SIMPURGEBLOCKEDJOBS	Especifica si moab debería eliminar de la simulación trabajos que no se van a poder ejecutar nunca
SIMRESOURCETRACEFILE	Especifica el fichero de recursos de la simulación
SIMSTARTTIME	Especifica en que instante de tiempo comenzará la simulación
SIMSTOPOPERATION	Especifica en que iteracion moab deberá parar para esperar comandos de usuario
SIMTIMEPOLICY	Especifica el paso de tiempo de forma real durante la simulación
SIMTIMERATIO	Determina el speedup del walltime
SIMWORKLOADTRACEFILE	Especifica el fichero de trazas de la simulación

Tabla 2.1: Parámetros de simulación del scheduler

2.8 Descripción de la carga

Los benchmark son programas que tienen como objetivo final medir las prestaciones de un sistema.
Estos programas no sólo pueden ayudarnos en la comparación de diferentes sistemas sino que además son capaces de evaluar las prestaciones de un equipo con diferentes configuraciones de Software y Hardware.

Mediante el resultado de la ejecución, se consigue estimar el rendimiento de un elemento concreto o de la totalidad del mismo.
El objetivo de los benchmark es que toda la comunidad utilice los mismos programas para medir sus sistemas, ya que es la única forma de poder comparar diferentes sistemas.

Los benchmark los podemos clasificar en diferentes tipos:

- *Instrucciones de adicción*: Fueron los primeros benchmark, y se caracterizan por ser programas de instrucciones reducidas donde las instrucciones más importante son de adicción. Actualmente están en desuso, puesto que debido a la evolución de los computadores, ya no son útiles.

- *Mezcla de instrucciones*: Benchmark que se caracterizan por ser una mezcla de instrucciones con su frecuencia de uso. Con el tiempo para cada instrucción, se puede calcular el tiempo medio para una mezcla de instrucciones dada. El benchmark más famoso en este grupo es Gibson.

- *Núcleos (kernels):* Debido a las optimizaciones de ejecución de instrucciones, el tiempo de instrucción puede ser muy variable, lo que hace que no podamos considerar las instrucciones de forma aislada, sino que debemos considerarlas conjuntos de instrucciones que forman una función de alto nivel. Las funciones más frecuentes se utilizan como carga de trabajo y reciben el nombre de kernel.

- *Programas sintéticos:* Son utilizados para medir la I/O (Input/output), el planteamiento más simple es un bucle que hace llamadas al subsistema de I/O. Estos tipos de benchmarks son desarrollados en su mayoría por lenguajes de alto nivel para aumentar su portabilidad.

Las políticas de planificación son muy sensibles al tipo de carga y longitud de la cola de trabajos, por ese motivo, se hace necesario utilizar diferentes tipos de Benchmarks, para poder estudiar como afectan esos factores a la política de planificación, ya que cada tipo de benchmark posee una carga de trabajo concreta dependiendo del tipo de prestación que se quiera medir.

2.8.1 NAS

Benchmark desarrollado por la NASA para evaluar las prestaciones de los sistemas paralelos distribuidos.
Los NAS Benchmark están compuestos de cuatro kernels paralelos:

- *EP (Embarrassingly Parallel):* Kernel encargado de medir el rendimiento del tratamiento de números en punto flotante.

- *MG (MultigridMethod):* Se encarga de medir el rendimiento de las comunicaciones de corto y largo alcance.

- *IS (Integer Sort):* Kernel que se encarga de medir el tiempo de operaciones con enteros y el rendimiento de las comunicaciones.

- *FT (FFT-based Spectral Method):* Kernel que se encarga de medir el rendimiento de las comunicaciones de larga distancia.

2.8.2 PARKBENCH

Benchmark que se utilizan para medir las prestaciones de computadores paralelos con memoria privada. Están escritos en Fortran77 con PVM/MPI.

Se dividen en los siguientes tipos:

- Programas en Fortran77 con PVM/MPI.
- Micro códigos: Parámetros arquitecturales básicos.
- Kernels: Subrutinas comunes en computación científica.
- Aplicaciones compactas: Aplicaciones NPB más un modelo espectral (PSTSWM).
- Códigos HPF: Aplicaciones sintéticas escritas en HPF.

2.8.3 Livermore

Benchmark desarrollado por el Laboratorio Nacional de Livermore (California), está compuesto por 24 kernels de aplicaciones numéricas de coma flotante, escritas en C y Fortran.
Son utilizados para comprobar el rendimiento en aritmética flotante tanto de escalares como de vectoriales.
La medida del rendimiento es dada en Millones de operaciones de punto flotante por segundo (MFLOPS).

2.8.4 LMBENCH

Benchmark portable para medir el estado latente, anchura de banda, costes de sistema operativo y transferencias de datos entre distintos componentes del sistema (procesador, cache, memoria principal, red, disco, etc). Diseñado únicamente para sistemas Unix.
La medida de rendimiento de este benchmark son las llamadas al sistema y las operaciones de movimiento de datos en UNIX

Capítulo 3

Selección del entorno de desarrollo

3.1 Introducción

En este capítulo presentamos los entornos de desarrollo propuestos en este estudio y explicamos la metodología de trabajo en este tipo de entornos.
Las configuraciones elegidas para los entornos de desarrollo han sido el resultado de un amplio estudio, tras el cual nos hemos decantado por las que más se han ajustado a nuestras necesidades.
El lector, una vez estudiado el capítulo anterior y visto los diferentes componentes, no tiene porque seleccionar la mismas configuraciones escogidas por nosotros, sino la que mejor se ajuste a sus necesidades.
En este capítulo sólo se pretende poner en práctica lo expuesto en el capítulo anterior, y explicar la metodología de trabajo.

3.2 Configuraciones específicas del entorno de desarrollo

Para nuestro estudio experimental hemos seleccionado dos entornos de desarrollo diferentes.
El primer entorno de desarrollo está basado en un cluster físico, mientras que el segundo se basa en el concepto de simulación.

3.2.1. Primer entorno de desarrollo

Para explicar el primer entorno de desarrollo, nos basaremos en la figura 3.1; que muestra como es la configuración de nuestro cluster; y a partir de ella explicaremos su configuración en detalle.

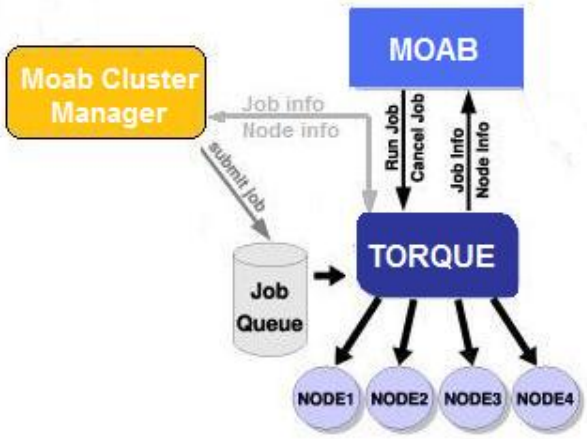


Figura 3.1: Configuración cluster físico

En primer lugar, describiremos el gestor de colas. Como se puede apreciar en la figura 3.1, el gestor de colas seleccionado es TORQUE.
Se optó por seleccionar este gestor principalmente por tres motivos. El primero de ellos es que es un gestor de colas avanzado y que está empezando a sustituir a PBS, con lo cual pensamos conveniente testarlo y comprobar de primera mano sus resultados.
El segundo motivo es que permite la integración con planificadores externos más avanzados que el que incorpora por defecto el propio TORQUE.
En último lugar, nos hemos decantado por seleccionar este gestor de colas debido a que permite la integración con una aplicación gráfica para enviar trabajos y consultar las estadísticas, con lo cual hace mucho más fácil el manejo del cluster ya que no hay que consultar al cluster vía consola y resulta mucho mas intuitivo.

Puesto que es el gestor de colas que hemos utilizado, consideramos importante explicar como es el gestor internamente, es decir, cuales son sus módulos principales y cual es su finalidad.
Básicamente, TORQUE está basado en 3 demonios o servicios, ya que se pueden configurar como demonio o como servicio indistintamente.
Estos 3 demonios son los siguientes:

- *pbs_ser*: Se ejecuta en el nodo servidor, con el cual iniciamos los servicios de Torque.

- *pbs_mon:* Se ejecuta en cada nodo del cluster para poder interactuar con el nodo servidor en el envió y recepción de trabajos.

- *pbs_sched:* Lanza el scheduler básico de Torque. En nuestro caso no lo hemos utilizado puesto que hemos preferido utilizar MOAB.

Torque sigue el mismo esquema de funcionamiento que PBS; utiliza los mismos demonios; la única diferencia entre ellos es la lógica interna que hace que Toque sea un gestor de colas bastante más avanzado que PBS.
Es necesario que, para que pbs_ser y pbs_mon puedan interactuar entre ellos, los nodos del cluster se encuentren todos bajo el mismo dominio.

Una vez visto el gestor de recursos pasaremos a explicar el planificador externo utilizado.
El planificador utilizado, como ilustra la figura 3.1, ha sido MOAB. Nos decantamos por utilizar este planificador externo puesto que es uno de los más potentes que hay actualmente y para el segundo entorno de desarrollo seleccionado nos resulta muy beneficioso y adecuado este planificador ya que posee la opción de trabajar en modo de simulación.
En caso de no haber querido utilizar el entorno de simulación podríamos haber utilizado el planificador que incorpora TORQUE por defecto, puesto que no estamos aprovechando todas las capacidades de MOAB, y no nos hubiera echo falta utilizar un planificador externo tan sofisticado para lanzar nuestras aplicaciones.
Puesto que en el capítulo anterior se explicó con detalle este planificador y en el siguiente capítulo se explicarán los scripts de configuración de MOAB, lo único que comentaremos en éste apartado es que como políticas de planificación hemos utilizado la FIFO y la BackFill. Se ha optado por utilizar estas dos políticas puesto que se ha pretendido comprobar la diferencia entre utilizar una política de estado actual frente a una política de estado futuro.

En cuanto al último componente utilizado para esta primera configuración, se trata de Moab Cluster Manager. Desde el principio del proyecto se tuvo claro que se pretendía trabajar en modo gráfico por su facilidad de uso y las ventajas que eso supone, ya que podemos lanzar trabajos o consultar estadísticas sin tener que escribir ningún script de configuración.
Existen diversas interfaces gráficas para el manejo de cluster, pero ninguna de ellas es tan avanzada como Moab Cluster Resource. Para argumentar la utilización de Moab Cluster Manager en nuestra configuración de trabajo, daremos un par de pinceladas a otra interface gráfica como es la interface de LoadLeveler.

Como podemos observar en la figura 3.2, el entorno gráfico de LoadLeveler es muy rudimentario. Se ejecuta en modo consola, lo que hace que actualmente esté obsoleto. En su salida al mercado, llegó a verse como un gran adelanto en cuanto a interficies gráficas de usuario para el manejo de clusters, pero debido a la reducida usabilidad que proporcionaba no llegó a prosperar.

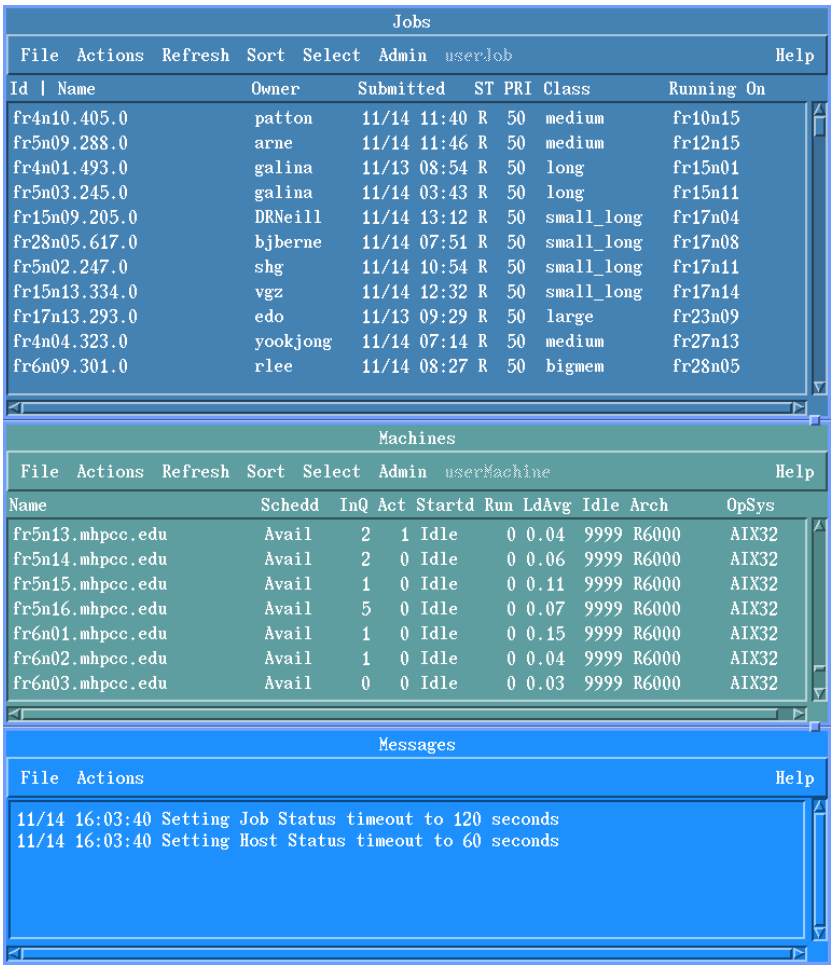


Figura 3.2: Entorno gráfico LoadLeveler

Como podemos apreciar en la figura 3.3, la interficie gráfica utilizada en nuestro entorno de trabajo es totalmente independiente del modo consola y nos permite el manejo completo del cluster; aparte de estar programada en Java; lo que hace que gracias a su máquina virtual podamos usarla tanto en máquinas con sistemas operativos GNU/Linux como en cualquier otro tipo de sistemas operativos.
Su apariencia amigable hace que un usuario medio pueda lanzar trabajos al cluster e interpretar el estado del cluster y de los trabajos de forma sencilla.

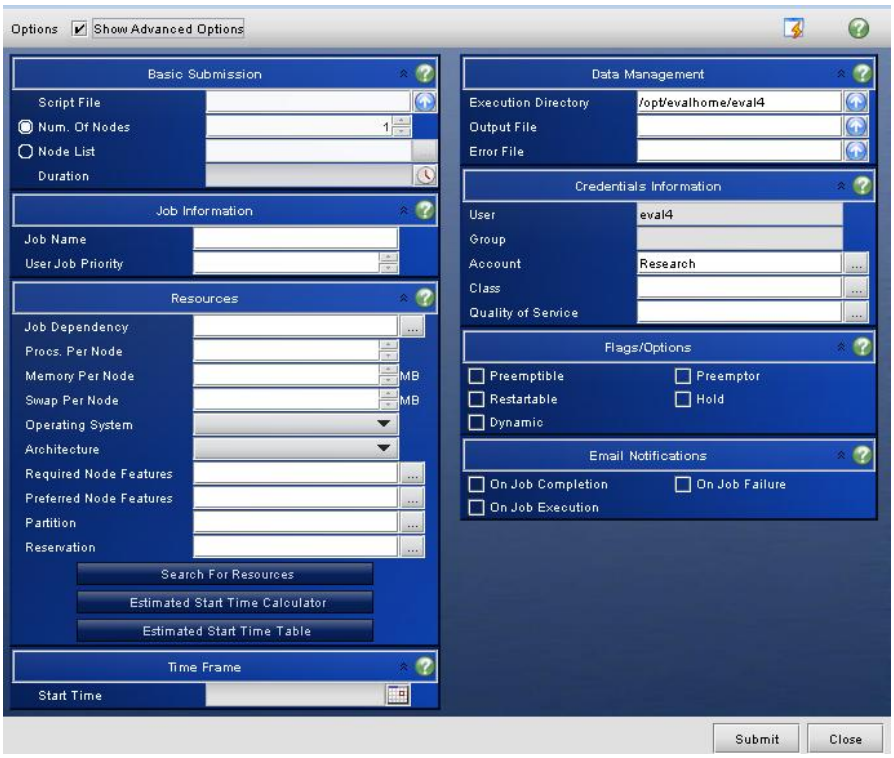


Figura 3.3 :Entorno gráfico Cluster Manager

Explicado y configurado el entorno de desarrollo se procederá a explicar la metodología de trabajo para este tipo de entornos.
La figura 3.4 muestra la pantalla principal de la interficie gráfica utilizada en este estudio. Como puede apreciarse en la figura, la pantalla principal nos muestra información general acerca del scheduler, nodos de cómputo, trabajos lanzados en el cluster e información del usuario que está utilizando la interficie.

Figura 3.4 : Pantalla principal de la interface gráfica

La figura 3.5 muestra la interface gráfica utilizada para el envío de trabajo al cluster, dicha interface nos ofrece todos los parámetros de configuración necesarios para la correcta ejecución de nuestro trabajo (número de nodos requeridos, número de procesadores por nodo, cantidad de memoria RAM, cantidad de memoria SWAP, selección del sistema operativo, selección de la arquitectura requerida, etc). Una vez rellenados todos los campos necesarios podemos enviar el trabajo a la cola de ejecución para ser ejecutado en el cluster.

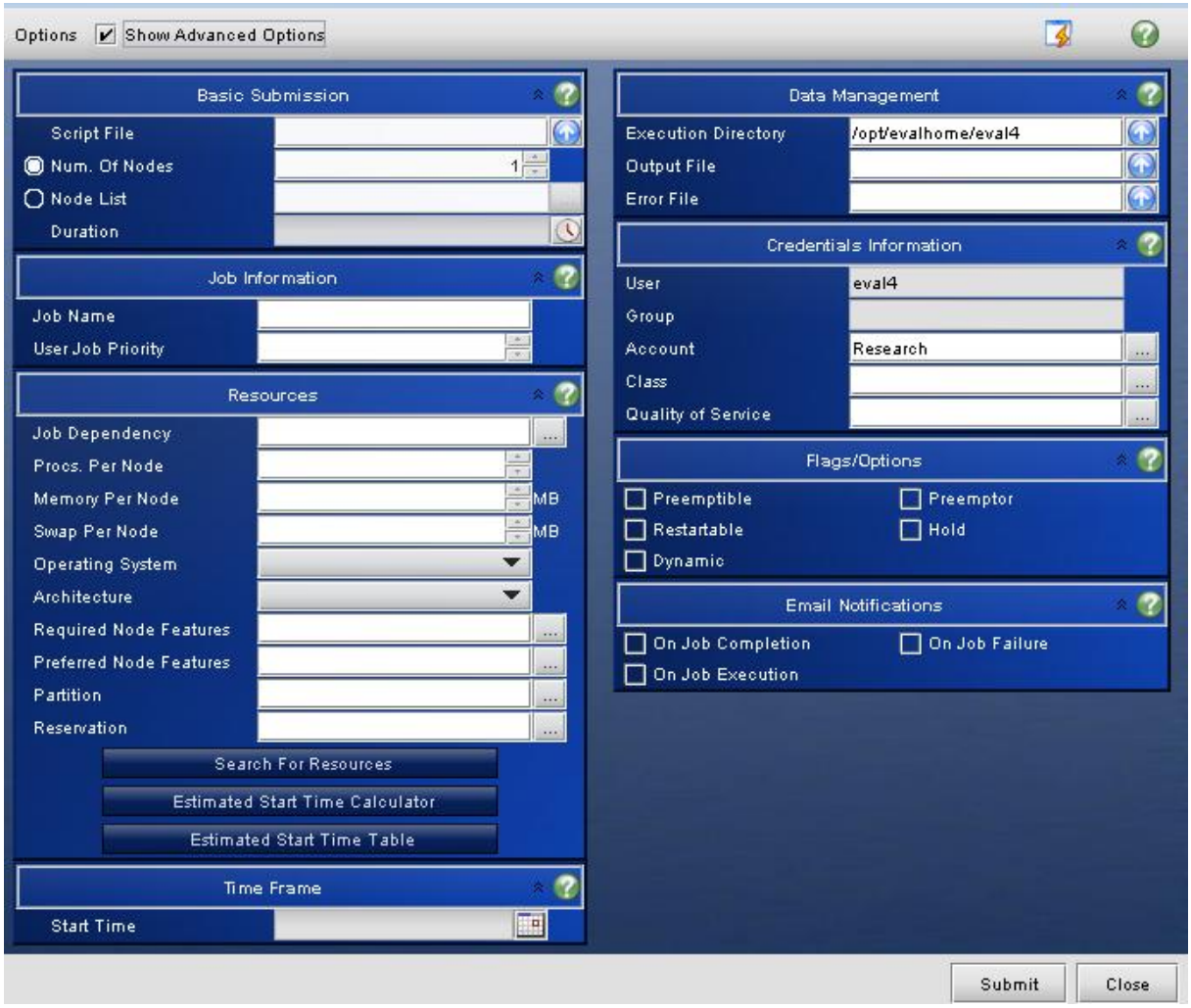


Figura 3.5: Pantalla para el envío de trabajos al cluster

Una vez lanzados los trabajos al cluster, podemos consultar el estado de nuestros trabajos en tiempo real, gracias a las herramientas de monitorización del estado del cluster, que nos ofrece la interficie gráfica. Dichas herramientas de monitorización, aparte de mostrarnos el estado de los trabajos disponibles en el cluster, también nos proporcionan información acerca del estado y la arquitectura de los nodos, de los recursos requeridos por cada trabajo, etc.
En las figuras 3.6 y 3.7 observamos como nos proporciona la interficie gráfica información acerca de los nodos de cómputo y de los trabajos.

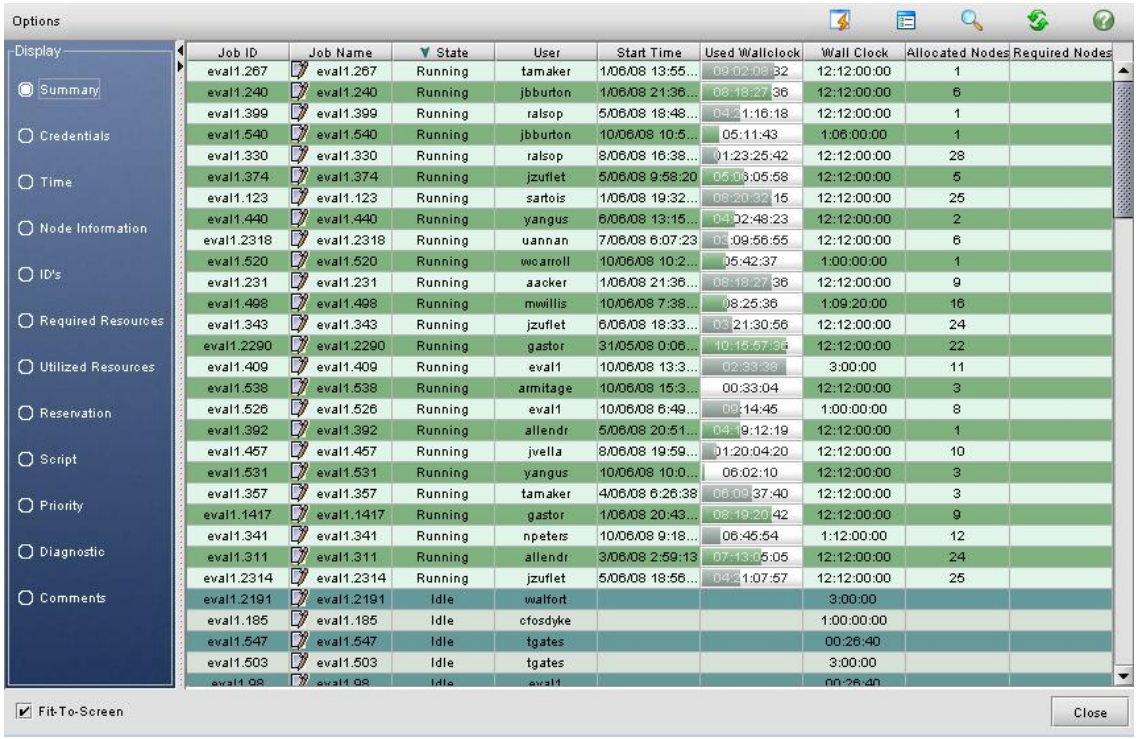


Figura 3.6: Información detallada de los trabajos del cluster

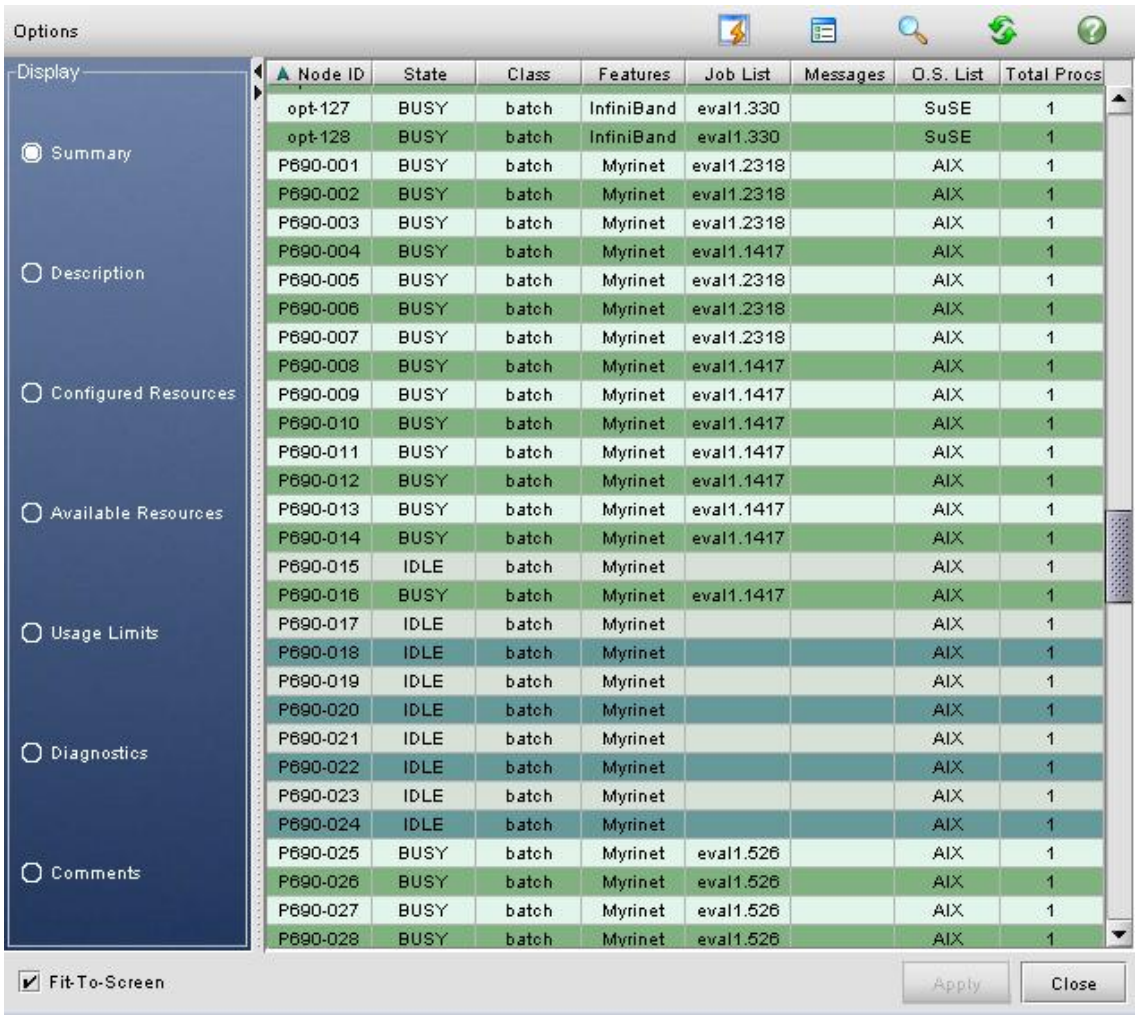


Figura 3.7: Información detallada acerca de los nodos del cluster

Una vez ejecutados los trabajos en el cluster, no sólo es importante conocer el resultado y el tiempo de cómputo del trabajo sino que también resulta importante conocer los recursos que ha consumido el trabajo durante su ejecución. La interface gráfica nos proporciona un extenso conjunto de herramientas de monitorización de recursos, que nos permiten conocer los recursos consumidos durante la ejecución de cualquier trabajo.
La figura 3.8 nos muestra los recursos consumidos de cpu para la ejecución de un determinado trabajo.

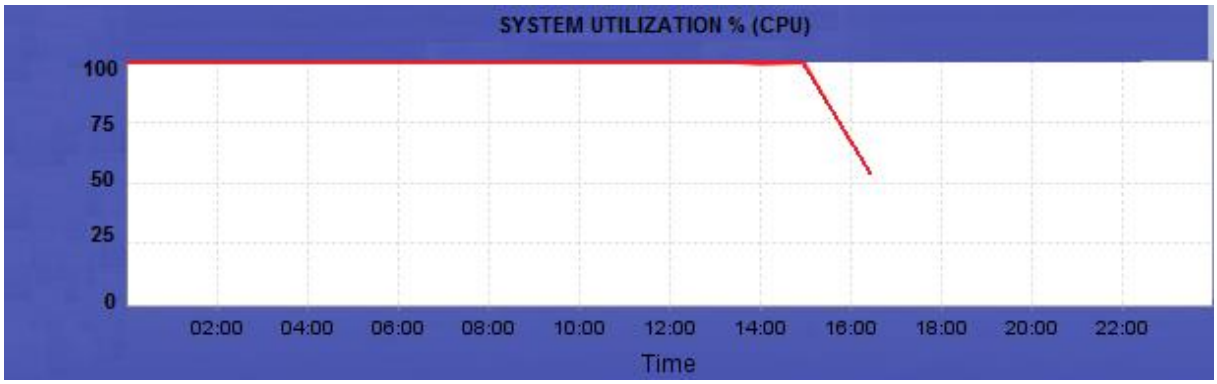


Figura 3.8: utilización de cpu durante la ejecución de un trabajo

3.2.2 Segundo entorno de desarrollo

Al igual que con el primer entorno, se mostrará una figura ilustrativa y se procederá a explicar sus componentes.

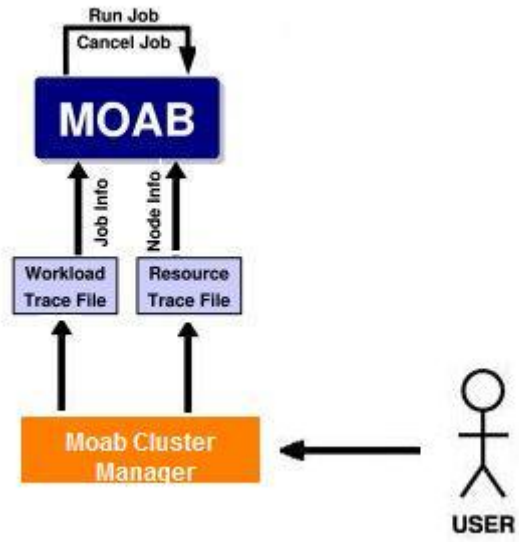


Figura 3.9: Segunda configuración seleccionada

Como podemos observar en la figura 3.9, el segundo entorno de desarrollo seleccionado obedece a una configuración claramente de simulación.
Como podemos ver en la figura, desaparece el gestor de colas y la entrada de trabajos por parte del usuario por los archivos de trazas.

El simulador recibe dos archivos de trazas, uno donde están especificados los recursos (Resource Trace File) y otro donde se especifica la carga a simular (Workload Trace File).

El fichero de recursos, especifica los recursos físicos del cluster; cada línea del fichero está compuesta por 21 campos obligatorios y representa un nodo simple. En la tabla 3.1 podemos ver los diferentes campos por los que está compuesto el fichero de recursos y su significado.

Campo	Significado
Tipo de recurso	El único valor posible actualmente para el campo es COMPUTENODE.
Tipo de evento	Especifica el estado del nodo.
Tiempo de evento	Actualmente este campo es ignorado por el simulador.
ID del recurso	Especifica el nombre del nodo.
Nombre del gestosr de recursos	Especifica el nombre del gestor de recursos con el que es asociada la simulación.
Configuración swap	Cantidad de memoria virtual asociada en el nodo.
Configuración memoria	Cantidad de memoria física asociada al nodo.
Configuración de disco	Cantidad de espacio local disponible en disco para los trabajos.
Configuración de procesadores	Número de procesadores disponibles en el nodo.
Configuración rack	Número de racks que dispone el nodo.
Localización del slot	Número del primer spot usado por el nodo (sólo SP2).
Cantidad de slots usados	Número de slots usados por el nodo.
Sistema operative	Sistema operativo utilizado en el nodo.
Arquitectura del nodo	Arquitectura del nodo.
Características del nodo	Lista de características o atributos del nodo.
Configuración de las clases de ejecución	Lista de parejas de la forma [CLASSNAME:CLASSCOUNT].
Adaptadores de red	Especifica el tipo de red utilizada para las comunicaciones.
Velocidad relativa de los recursos	Velocidad relativa de la máquina. Este campo tiene valor 1 por defecto.
Campo	Campo no usado por el usuario.

reservado	
Campo reservado	Campo no usado por el usuario.
Campo reservado	Campo no usado por el usuario.

Tabla 3.1: Parámetros del fichero de recursos

Los campos de la tabla 3.1 son obligatorios a la hora de construir el archivo de recursos y deben aparecer en el orden mostrado en la tabla. Una alteración del orden provocaría que el fichero de recursos fuera incorrecto. En caso de querer obviar un campo, este se especifica mediante un guión "-". En la figura 3.10 podemos ver un ejemplo de un archivo de recursos tal y como se especificaría.

```
#Nodo 1
COMPUTENODE AVAILABLE 0 nodo1 PBS 423132 256 7140 2 -1 -1 1 LINUX62
AthlonK7 [s950][compute]  [batch:2]  [ethernet] 1.67  -  -  -

#Nodo 2
COMPUTENODE AVAILABLE 0 nodo2 PBS 423132 512 7140 2 -1 -1 1 LINUX62
AMD [s950][compute]  [batch:1]  [ethernet] 1.67  -  -  -
```

Figura 3.10: Ejemplo fichero de recursos

El archivo de trazas describe el conjunto de trabajos que serán simulados y todos los aspectos relevantes del scheduling. También especifica el entorno de ejecución de cada trabajo. Cada línea del fichero especifica un evento diferente (**JOBSUBMIT**, **JOBSTART**, **JOBEND**). El archivo puede contener todos los trabajos que queramos simular siempre y cuando se respete el orden de los eventos para cada trabajo.

En la tabla 3.2 podemos ver los campos del archivo de trazas:

Campo	Significado
Evento de tiempo	Especifica cuando ocurrirá el evento (HH:MM:SS).
Evento de tiempo(En formato EPOCH)	Especifica cuando ocurrira el evento(formato epochtime).
Tipo de objeto	Especifica el tipo de objeto.
ID del obejto	Especifica la identidad del objeto.
Evento del objeto	Especifica el tipo de evento del objeto.
Nodos requeridos	Especifica el conjunto de nodos requeridos para ejecutar el trabajo.
Tareas requeridas	Número de tareas requeridas.
Nombre de usuario	Especifica el nombre del usuario que ha subministrado el trabajo.
Nombre de grupo	Especifica el nombre del grupo al que pertenece el usuario que ha subministrado el trabajo.
Limite de Wallclock	Límite máximo de la duración de un trabajo.
Estado del evento	Estado del trabajo cuando se produce el evento.
Tiempo de subministro	Especifica cuando se subministra el trabajo al scheduler (Formato EPOCH).
Tiempo de	Especifica cuando el scheduler empieza a ejecutar el trabajo (Formato

aceptación	EPOCH).
Tiempo de ejecución	Especifica cuando el scheduler empieza a ejecutar el trabajo (Formato EPOCH), el valor debe de ser igual al tiempo de aceptación.
Tiempo de terminación	Especifica el tiempo en el que el trabajo se ha completado (Formato EPOCH).
Adaptador de red	Especifica el tipo de red necesario para ejecutar el trabajo.
Arquitectura del nodo	Especifica la arquitectura del nodo que necesita el trabajo para poder ejecutarse.
Sistema operative	Especifica el sistema operativo necesario para ser ejecutado un trabajo.
Comparación de memoria requerida	Se utiliza para comparar los requisitos de memoria necesarios para ejecutar el trabajo en el nodo.
Memoria requerida	Cantidad de memoria requerida en cada nodo.
Comparación de disco	Se utiliza para comparar los requisitos de disco necesarios para ejecutar el trabajo en el nodo.
Disco requerido	Cantidad de disco requerido en cada nodo para ejecutar el trabajo.
Características del nodo	Se utiliza para determinar las características que tiene que cumplir el nodo para poderse ejecutar el trabajo.
Ejecutable	Nombre del trabajo ejecutable si se ha especificado.
Procesadores temporalmente utilizados	Número de procesadores temporalmente utilizados por el trabajo.
Nombre de la partición	Nombre de la partición donde se ejecuta el trabajo.
Procesadores dedicados por tarea	Número de procesadores dedicados por tareas.
Memoria dedicada por tarea	Cantidad de memoria RAM por tarea.
Disco necesario por trabajo	Cantidad de espacio requerido en disco por tarea.
Memoria virtual dedicada	Cantidad de memoria virtual necesaria por cada tarea.
Fecha de comienzo	Indica la fecha más temprana en la que puede empezar a ejecutarse un trabajo.
Fecha de terminación	Indica la fecha donde todos los trabajos tienen que estar ejecutados.
Gestor de recursos	Nombre del gestor de recursos que utilizamos para ejecutar el trabajo.
Reservas	Nombre de la reserva requerida para ejecutar un trabajo.
History	Guarda el historico de recursos de simulaciones anteriores
Codigo complete	Este campo no es utilizado actualmente por el simulador.
Información extra de memoria	Extiende las estadísticas de uso de la memoria.
Información extra de CPU	Extiende las estadísticas de uso de la CPU.

Tabla 3.2: Parámetros del fichero de trazas

Como sucede con el fichero de recursos, los campos de la tabla anterior son obligatorios a la hora de construir el archivo de recursos y deben aparecer en el orden mostrado en la tabla. Una alteración del orden provocaría que el fichero de recursos estuviera incorrecto. En caso de querer obviar un campo, este se especifica mediante un guión "-". En la figura 3.11 podemos ver un ejemplo de un archivo de recursos tal y como se especificaría.

```
13:21:05 110244355 job 1413 JOBEND 20 20 josh staff 86400 Removed [batch:1] 887343658 889585185 \
889585185 889585411 ethernet R6000 AIX53 >= 256 >= 0 - 889584538 20 0 0 2 0 test.cmd \
1001 6 678.08 0 1 0 0 0 0 0 - 0 - - - - - - - - - 0.0 - - - 0 - -
```

Figura 3.11: Ejemplo de un fichero de trazas

Como hemos podido ver a lo largo de toda la explicación, el trabajo en este tipo de entornos es bastante complicado debido a todos los parámetros que hemos de considerar en los archivos de trazas que hay que subministarle al simulador, lo que hace que el usuario deba tener unos conocimientos bastante avanzados para poder obtener prestaciones adecuadas.

Una vez construidos los archivos de trazas, podemos proceder a ejecutar la simulación. Mediante la utilización de la interface gráfica, podemos configurar el scheduler para que trabaje en modo de simulación.
La figura 3.12 muestra los diferentes modos de trabajo con los que podemos configurar el scheduler.

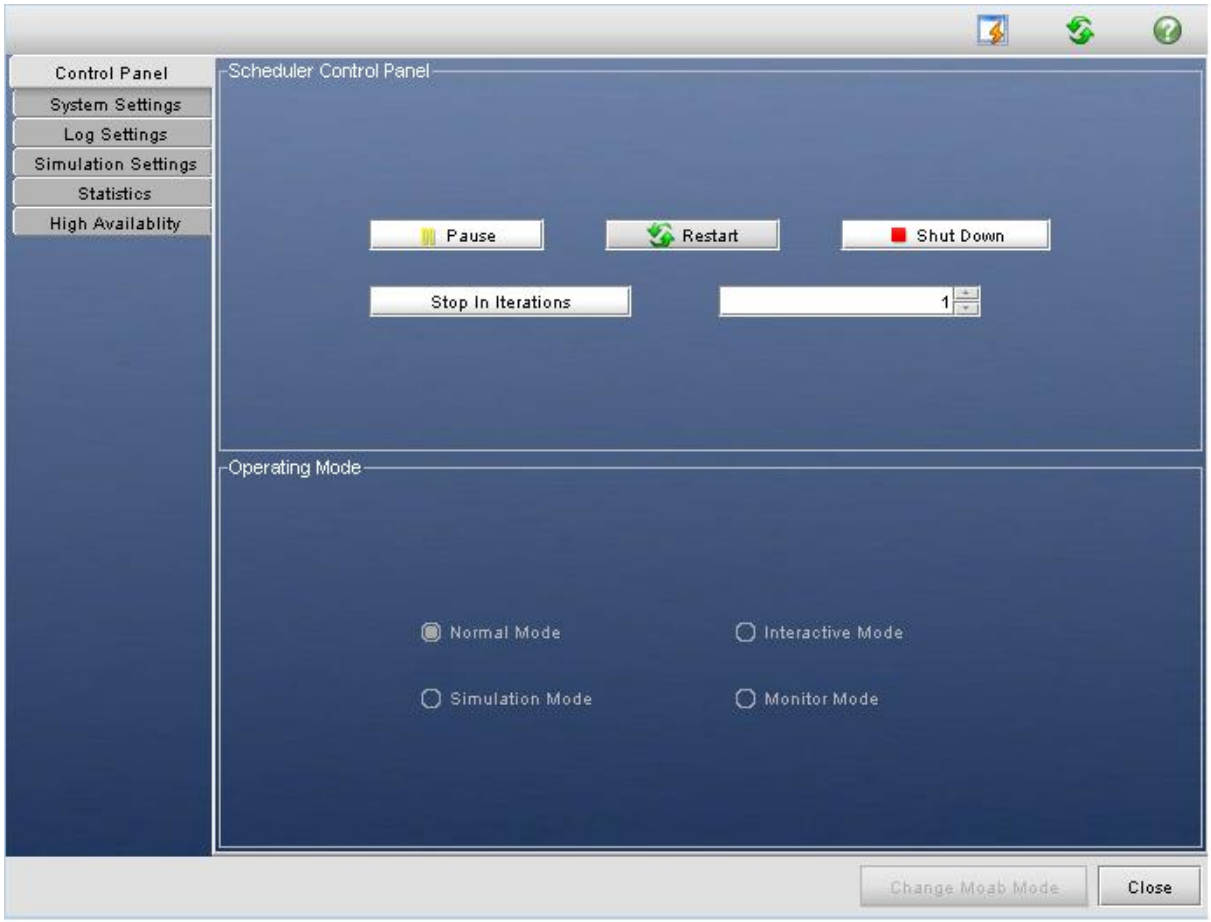


Figura 3.12: Inteficie de configuración del modo de trabajo del scheduler

Una vez hemos seleccionado el modo de simulación, en la izquierda de la figura 3.12 observamos que tenemos una pestaña llamada “Simulations Settings”. Seleccionando dicha pestaña, nos muestra un menú donde podemos configurar algunos parámetros de la simulación.

La figura 3.13 nos muestra el menú de configuración de los parámetros de la simulación.

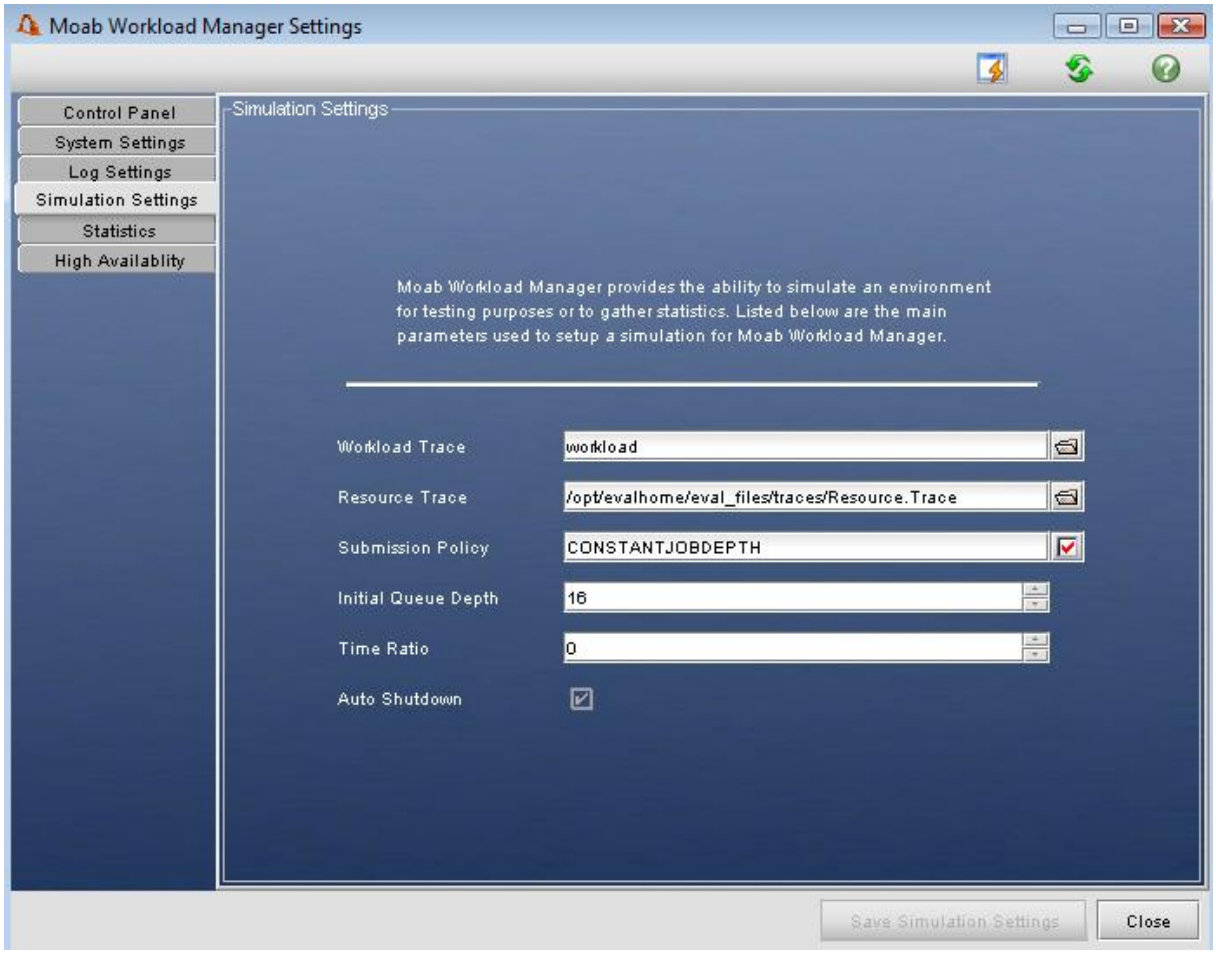


Figura 3.13: Interficie de configuración de los parámetros de simulación

Como podemos observar en la figura 3.13, la interficie gráfica nos permite proporcionarle el archivo de trazas y el archivo de recursos necesario para poder ejecutar la simulación. También nos permite subministrarle la política de planificación a utilizar y la carga de trabajos inicial a la cola de ejecución, así como el Ratio del tiempo.
El resto de parámetros de simulación los inicializa la interficie gráfica por defecto. En caso de querer cambiar el valor de algún parámetro de simulación que no este disponible en la interfice gráfica, se hace necesario modificar el archivo de configuración de Moab "moab.cfg".
Como se argumentó en el segundo capítulo, en el apartado dedicado a los simuladores, existe un amplio abanico de parámetros destinados a simulación, lo que hace que éste sea un proceso muy complicado y requiera conocimientos avanzados sobre el scheduler. En la tabla 3.3 podemos ver los principales parámetros de simulación. Para obtener la lista completa, el lector puede acudir al anexo 6 de este mismo trabajo.

Parámetro	Descripción
SIMULATIONSHUTDOWN	La simulación se termina cuando la cola queda vacía
SIMCPUSCALINGPERCENT	Especifica si incrementar o decrementar el reloj del runtime de cada trabajo

SIMIGNOREJOBFLAGS	Especifica que atributos o criterios ignorará el simulador del fichero de trazas
SIMINITIALQUEUEDEPTH	Especifica cuantos trabajos pueden estar en la cola de ejecución con estado IDLE durante la simulación
SIMJOBSUBMISSIONPOLICY	Especifica como el simulador subministra nuevos trabajos a la cola durante la simulación
SIMNODECOUNT	Especifica el máximo número de nodos que pueden utilizarse para la simulación
SIMPURGEBLOCKEDJOBS	Especifica si moab debería eliminar de la simulación trabajos que no se van a poder ejecutar nunca
SIMRESOURCETRACEFILE	Especifica el fichero de recursos de la simulación
SIMSTARTTIME	Especifica en que instante de tiempo comenzará la simulación
SIMSTOPOPERATION	Especifica en que iteracion moab deberá parar para esperar comandos de usuario
SIMTIMEPOLICY	Especifica el paso de tiempo de forma real durante la simulación
SIMTIMERATIO	Determina el speedup del walltime
SIMWORKLOADTRACEFILE	Especifica el fichero de trazas de la simulación

Tabla 3.3: Parámetros de simulación

Capitulo 4

Evaluación experimental

4.1 Introducción

El presente capítulo expone la forma de trabajar en este tipo de entornos, basándose en ejecuciones reales sobre el cluster, a fin de poder mostrar las herramientas que nos ofrece la interface gráfica de una forma práctica. Se ha dividido el capítulo de forma que quede claramente diferenciada la parte de desarrollo en entorno real de la simulada.
El objetivo que se pretende alcanzar en este capítulo es minimizar la dificultad que supone trabajar en este tipo de entornos debido al gran número de parámetros que hay que conocer para poder sintonizar correctamente la ejecución de las aplicaciones; y cómo la mayoría de las veces, el desconocimiento del comportamiento de las aplicaciones nos lleva a tener que utilizar la simulación, para intentar sintonizar las aplicaciones antes de lanzarlas al cluster.
En el presente capítulo se exponen las herramientas de monitorización que nos ofrece la interficie gráfica, como instrumento de ayuda para intentar entender el comportamiento de nuestras aplicaciones, a fin de poder sintonizarlas correctamente.

4.2 Ejecución en entorno real

Para la ejecución de las aplicaciones en el cluster hemos utilizado un entorno gráfico, como se ha mostrado en el capítulo anterior. La utilización de este tipo de entorno gráfico puede dar la impresión a primera instancia que cualquier usuario con conocimientos nulos en este campo pueda ejecutar aplicaciones en el cluster, debido a la facilidad que nos ofrece la interfaz a la hora de ejecutar trabajos. Si bien es cierta la afirmación previa, debido a la gran cantidad de parámetros que son necesarios subministrar al cluster para la ejecución de trabajos (como el gran número de parámetros de configuración que posee el scheduler), hace que un usuario que desconozca todos esos parámetros ejecute los trabajos de manera descontrolada y carezcan de utilidad.

El primer problema con el que nos encontramos a la hora de ejecutar trabajos en el cluster es el desconocimiento del comportamiento de la aplicación, que hace que desconozcamos los recursos necesarios que necesitará para ejecutarse.
Es muy importante intentar sintonizar los parámetros de ejecución a los recursos que verdaderamente necesitamos, debido a que una ejecución sobre un cluster real, tiene un elevado coste económico.

Para la ejecución de la parte práctica del proyecto hemos utilizado las aplicaciones IS de los NAS Benchmark y un programa de multiplicación de matrices(5000*5000) escrito utilizando la librería de paso de mensajes MPI. Debido a que nuestro cluster sólo dispone de un nodo, hemos ejecutado las aplicaciones en el cluster real sintonizando todos los parámetros necesarios para su ejecución a los recursos de nuestro cluster, y posteriormente en el apartado de simulación volveremos a simular la misma carga de trabajo para distintas configuraciones a fin de poder afinar la sintonización a los recursos que verdaderamente necesitamos.

Los recursos seleccionados para la ejecución de las aplicaciones en el cluster han sido los siguientes:

Recursos	Valor
Numero de nodos	1
Numero de cpu's por nodo	1
Memoria física	2048Mb
Red	Ethernet
Arquitectura del nodo	X86
Sistema operativo	Linux64
Memoria swap	250Mb
Wallclok	1:00:00

Tabla 4.1: Recursos necesarios para la ejecución de las aplicaciones

Para subministrar tanto las aplicaciones como los recursos necesarios lo haremos a través del menú que nos ofrece la interface gráfica. La figura 4.1 muestra la interfice gráfica con los campos de la tabla 4.1 rellenados.

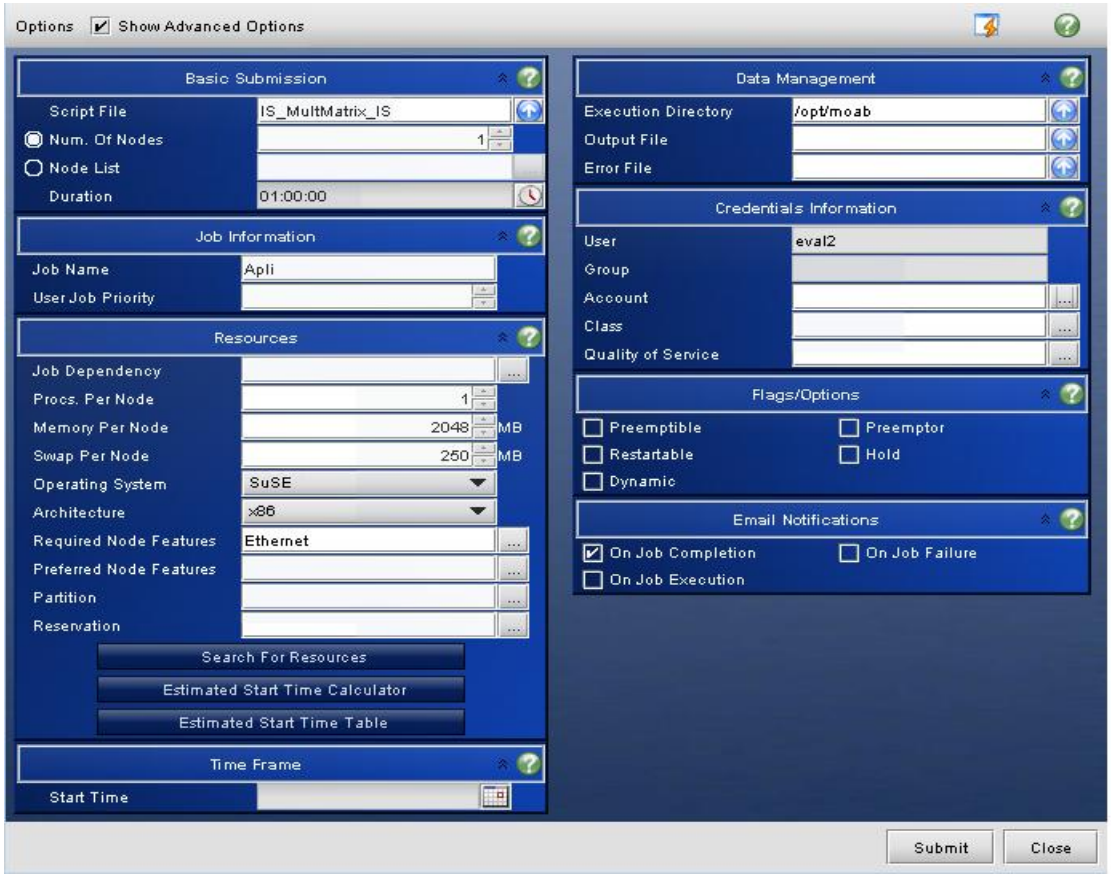


Figura 4.1: Interfice gráfica para el envío de trabajo al cluster

Puesto que estamos ejecutando sobre el cluster real, tenemos que adecuar los recursos necesarios a las prestaciones que nos ofrece nuestro cluster; si cuando subministramos un nuevo trabajo al cluster rellenamos la casilla de número de nodos con un valor superior a 1, que es el número de nodos que posee nuestro cluster, en la pantalla principal de la interface gráfica podemos observar que sólo un nodo aparece como busy, es decir, que está activo ejecutando trabajos, mientras que el resto aparecen en estado offline. La figura 4.2 ilustra lo explicado lanzando una ejecución sobre el cluster, donde hemos sintonizado a 6 el número de nodos.

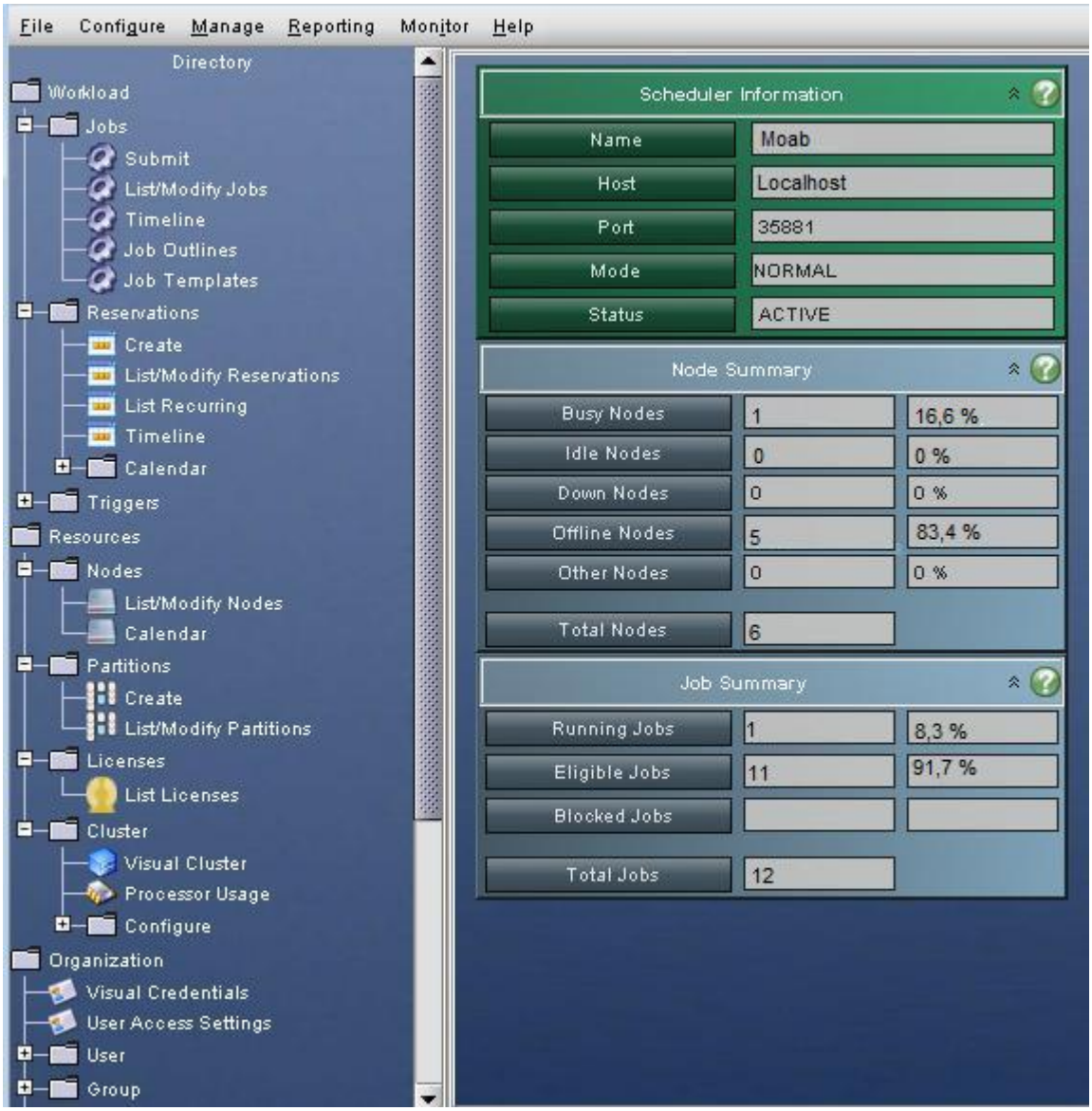


Figura 4.2: Pantalla que muestra el estado de los nodos

En la pantalla principal de la interface gráfica podemos consultar el estado de nuestros trabajos (Blocked, Elegible, Runinng). El conjunto de trabajos se habrá ejecutado cuando no queden trabajos pendientes en estado Elegible o Running.
Como podemos apreciar en la figura 4.3 el número total de trabajos que hay actualmente en el cluster es de 12. Puesto que nuestro cluster es de 1 solo nodo, encontraremos que se está ejecutando un único trabajo, mientras el resto permanecerá en estado Elegible hasta que el trabajo que se está ejecutando termine, para poder dejar paso a los otros trabajos.

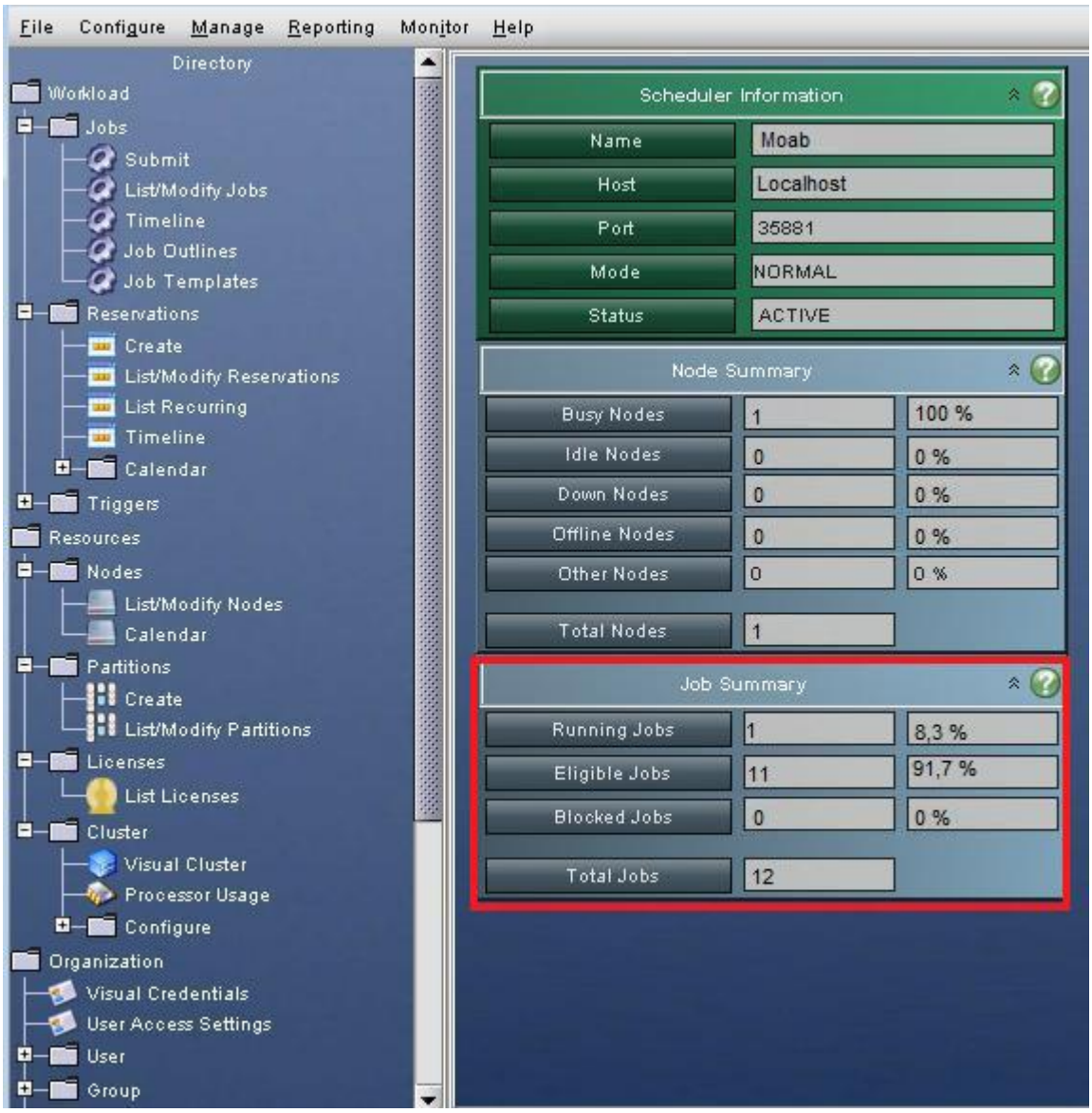


Figura 4.3: Estado de los trabajos en el cluster

La interficie gráfica, mediante las herramientas de monitorización, nos permite conocer los recursos utilizados durante la ejecución de las aplicaciones. Las figuras 4.4, 4.5 y 4.6 nos muestran los recursos de CPU, memoria y red utilizados para la ejecución realizada anteriormente.

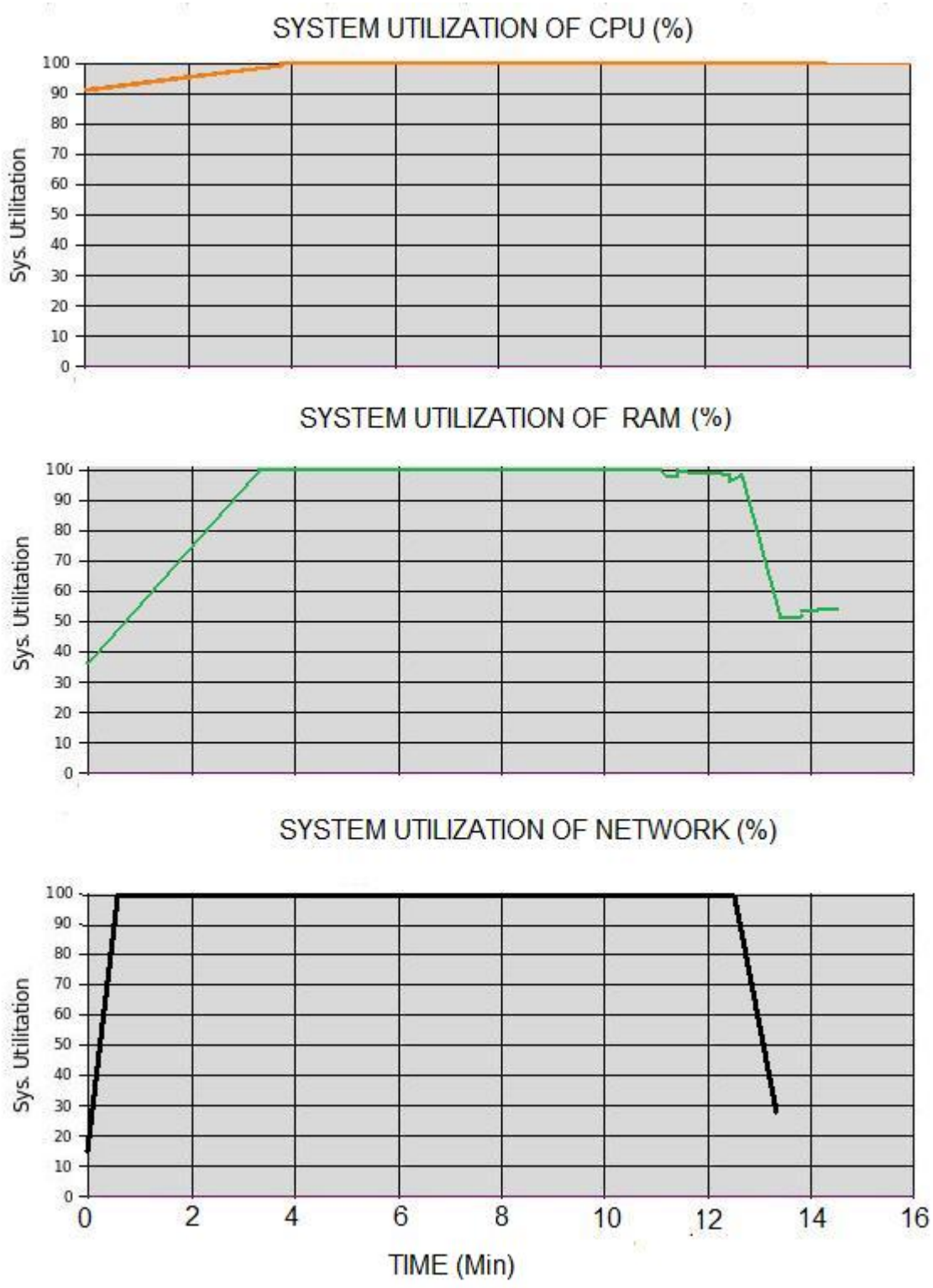


Figura 4.4: Gráficas de utilización de recursos durante la ejecución de los trabajos

Puesto que el objetivo de este capítulo no es discutir los resultados obtenidos de las ejecuciones, sino mostrar las herramientas que nos ofrece el entorno gráfico. Sólo comentaremos que observando las gráficas resulta fácil darse cuenta que los recursos están claramente saturados, lo que pone de manifiesto la complicación de afinar los recursos necesarios para la ejecución de trabajos, sin antes haber pasado por un periodo de simulación.

4.3 Ejecución en simulación

Es habitual en usuarios inexpertos pensar que el proceso de simulación es trivial; ya que si disponemos del simulador y de las aplicaciones que queremos simular; solamente debemos esperar a que el simulador nos dé los resultados. En la realidad es todo lo contrario, ya que la dificultad de la simulación reside en conocer la infinitud de parámetros que requiere y saber sintonizarlos correctamente, como se argumento en el capítulo anterior. Otro proceso realmente complicado es el de aceptación de los resultados, ya que puede llegar a ser verdaderamente complicado entender que es lo que está realmente haciendo la simulación en cada momento.
En el presente apartado se intenta poner de manifiesto las dificultades expuestas anteriormente para que el lector comprenda el verdadero grado de dificultad en el proceso de simulación.
Para intentar comprender los resultados de la simulación nos ayudaremos de las herramientas de monitorización que nos proporciona la interficie gráfica; y gracias a las cuales podemos ver y entender el comportamiento de la simulación.

Hemos simulado la misma carga de trabajo que en el entorno real, cambiando los argumentos de las aplicaciones IS de tipo A y B dependiendo del número de procesadores de la simulación. Se han aplicado diversas políticas de planificación y se han mostrado diferentes aspectos de los resultados de las simulaciones.

Para poder llevar a cabo la simulación, hemos optado por crear los ficheros de trazas de forma automática, pues Moab nos segura que, siempre y cuando hayamos antes ejecutado la carga que queramos simular en el cluster físico, nos creará una copia idéntica de nuestro cluster y de las aplicaciones a simular. Para crear los ficheros de trazas aprovecharemos la ejecución que hemos realizado anteriormente y crearemos los ficheros de trazas y recursos de la siguiente forma:

- *Resource.trace:* Lo hemos creado mediante el comando "schedctl –n > Resource.trace". A través de la sentencia anterior obtenemos los recursos de nuestro cluster. Redireccionando la salida del comando a un fichero conseguimos obtener el fichero de recursos del cluster físico.

- *Workload.trace:* Una vez la ejecución real ha finalizado, en el directorio stats de Moab se nos crea un fichero con todos los eventos que han sucedido en el planificador (comienzo de trabajos, finalización de trabajos, etc). Copiando ese fichero de la siguiente manera " cp stats/fichero_eventos workload.trace" obtenemos el fichero de trazas de la simulación.

Las figuras 4.5 y 4.6 ilustran un fragmento de los ficheros de trazas

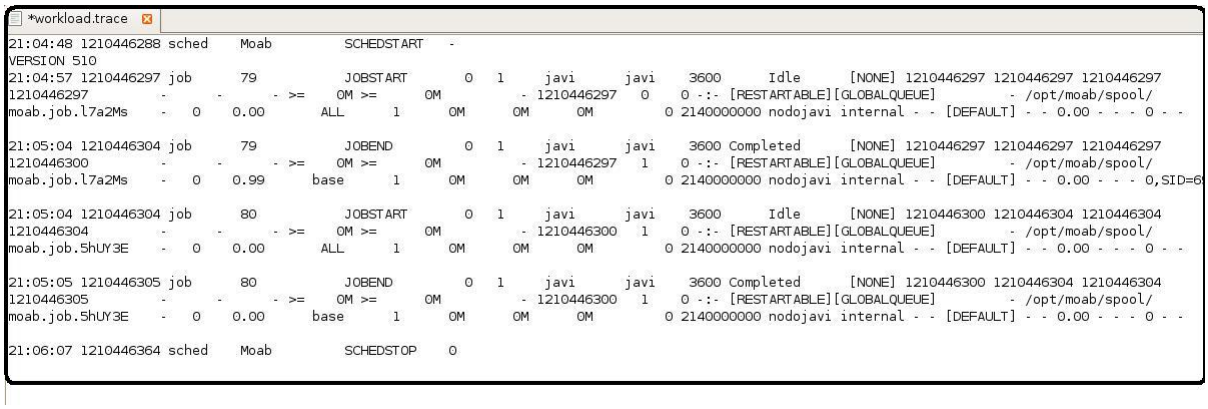


Figura 4.5: Archivo de trazas

Lo interesante de la figura 4.5 es observar como para poder simular las aplicaciones necesitamos proporcionarle todos los eventos necesarios que necesita la simulación. En la primera línea podemos ver como el primer paso a realizar es activar el scheduler, momento a partir del cual da comienzo la simulación. Acto seguido le proporcionamos los trabajos a simular; podemos observar que por cada trabajo a simular es necesario indicarle tanto el evento de comienzo como de finalización del trabajo. La última línea muestra la finalización del scheduler y por consiguiente de la simulación

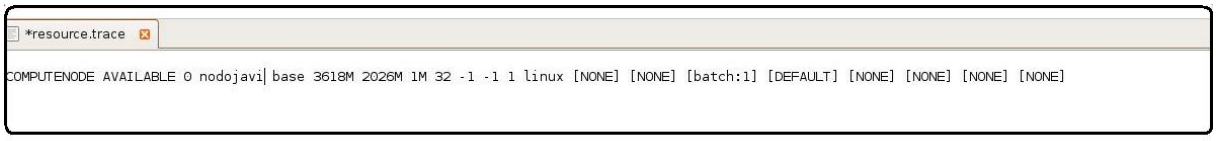


Figura 4.6: Archivo de recursos

La figura 4.6 muestra el archivo de recursos utilizado para simular las aplicaciones. Como podemos apreciar en la figura, utilizamos 3618Mb de memoria swap, 2026Mb de memoria RAM, 1MB de espacio en disco local y 32 procesadores como recursos para simular nuestras aplicaciones.

A partir de esos ficheros y viendo los resultados de las simulaciones, podemos ir modificando sus parámetros de forma manual a fin de poder llegar a obtener los parámetros óptimos de ejecución.

Como hemos podido demostrar, para que Moab nos permita crear los ficheros de trazas de forma automática es necesario lanzar una ejecución en el cluster físico; hecho que la mayoría de las veces no es posible; ya que desconocemos el comportamiento de la ejecución y no podemos parametrizar sus recursos a primera instancia sin antes haber estudiado la aplicación en el contexto de la simulación, puesto que la ejecución en el cluster físico tiene una repercusión elevada bastante importante, lo que supone que la mayoría de la veces tengamos que construir los archivos de trazas de forma manual.
Puesto que ejecutar nuestras aplicaciones en el cluster real no supone una gran repercusión, hemos optado por hacer los archivos de trazas de forma automática y posteriormente ir modificándolos de forma manual

Construidos los archivos de simulación, en el fichero de configuración de Moab tenemos que seleccionar y configurar los parámetros de la simulación (ver Anexo 5: Parámetro de

configuración de Moab). El entorno gráfico utilizado nos permite seleccionar los parámetros más importantes de la simulación, pero independientemente de los parámetros que nos permite seleccionar el entorno gráfico, existen muchos otros parámetros que es conveniente considerar para lanzar la simulación correctamente. Debido a la gran cantidad de parámetros disponibles en el scheduler que hacen referencia a la simulación, hace que este punto pueda resultar verdaderamente complicado, pues exige un conocimiento profundo del significado de todos los parámetros de simulación del scheduler y su interrelación entre ellos. La tabla 3.3 del capítulo anterior nos muestra los parámetros más importantes de simulación, para obtener la lista completa podemos consultar el anexo 6 de este trabajo.

Una vez tenemos preparado el simulador procedemos a simular el trabajo en 1, 2, 4, 8, 16 , 32 y 64 procesadores para las políticas de planificación FIFO y Backfill, y estudiaremos sus resultados mediante las herramientas de monitorización.

Para realizar la simulación hemos utilizado la interficie gráfica, ya que nos permite configurar el scheduler de forma rápida en modo de simulación y subministrarle los archivos de trazas.
La figura 4.7 muestra la pantalla donde le proporcionamos los archivos de trazas al scheduler. En dicha pantalla también podemos seleccionar la política de planificación a utilizar y el tamaño inicial de la cola de ejecución. En nuestro caso hemos seleccionado una política FIFO y hemos establecido el tamaño inicial de la cola de simulación a 12 trabajos.

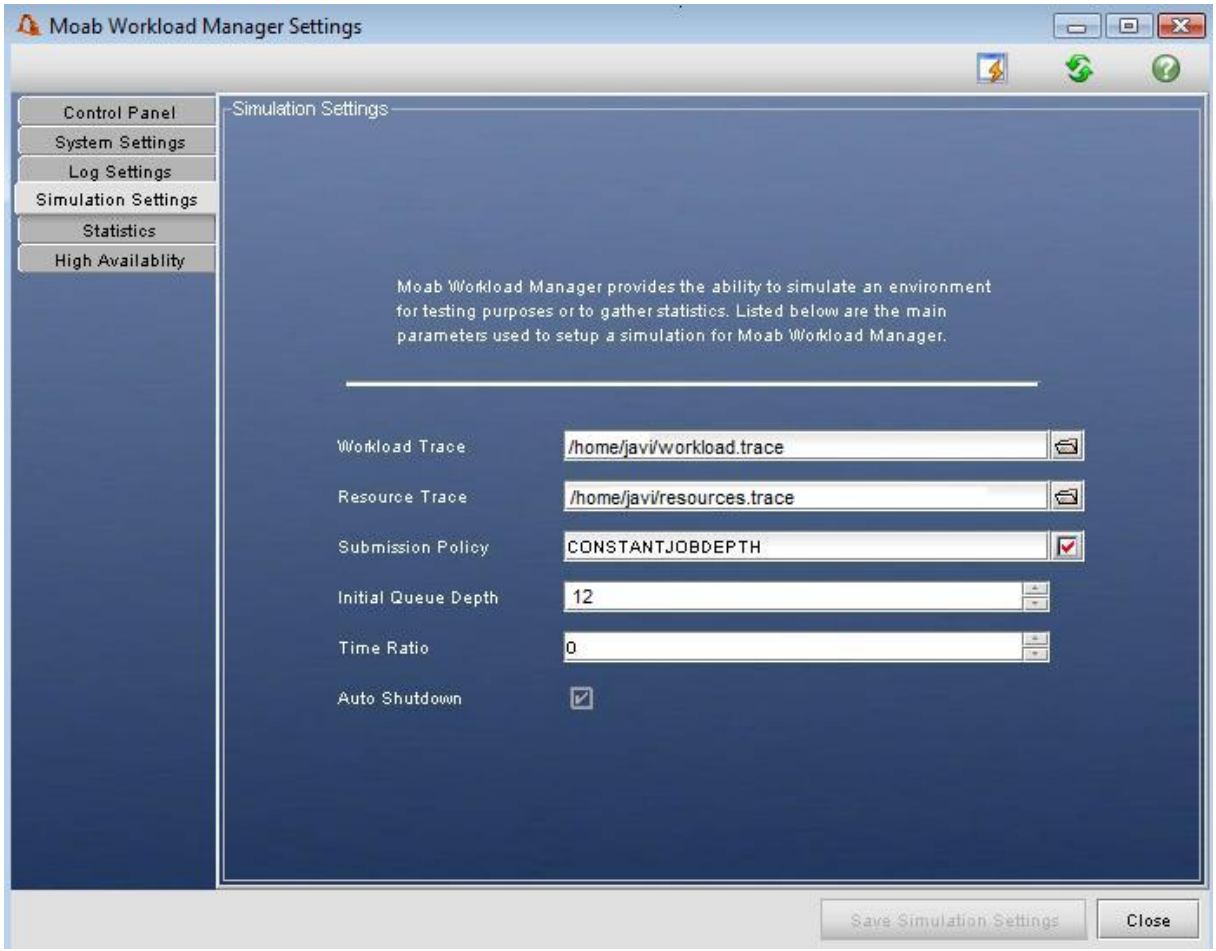


Figura 4.7: Configuración de los parámetros de simulación

Una vez tenemos el scheduler en modo de simulación podemos hacer diferentes simulaciones cambiando los parámetros de la simulación mediante los archivos de trazas.

Un punto importante a resaltar es que los archivos de trazas se tienen que modificar de forma manual, ya que la interficie gráfica no permite modificarlos.
La interficie gráfica permite conocer todos los resultados de la simulación gracias a las herramientas de monitorización.
Para la simulación en 1 procesador hemos obtenido los resultados mostrados en las figura 4.8

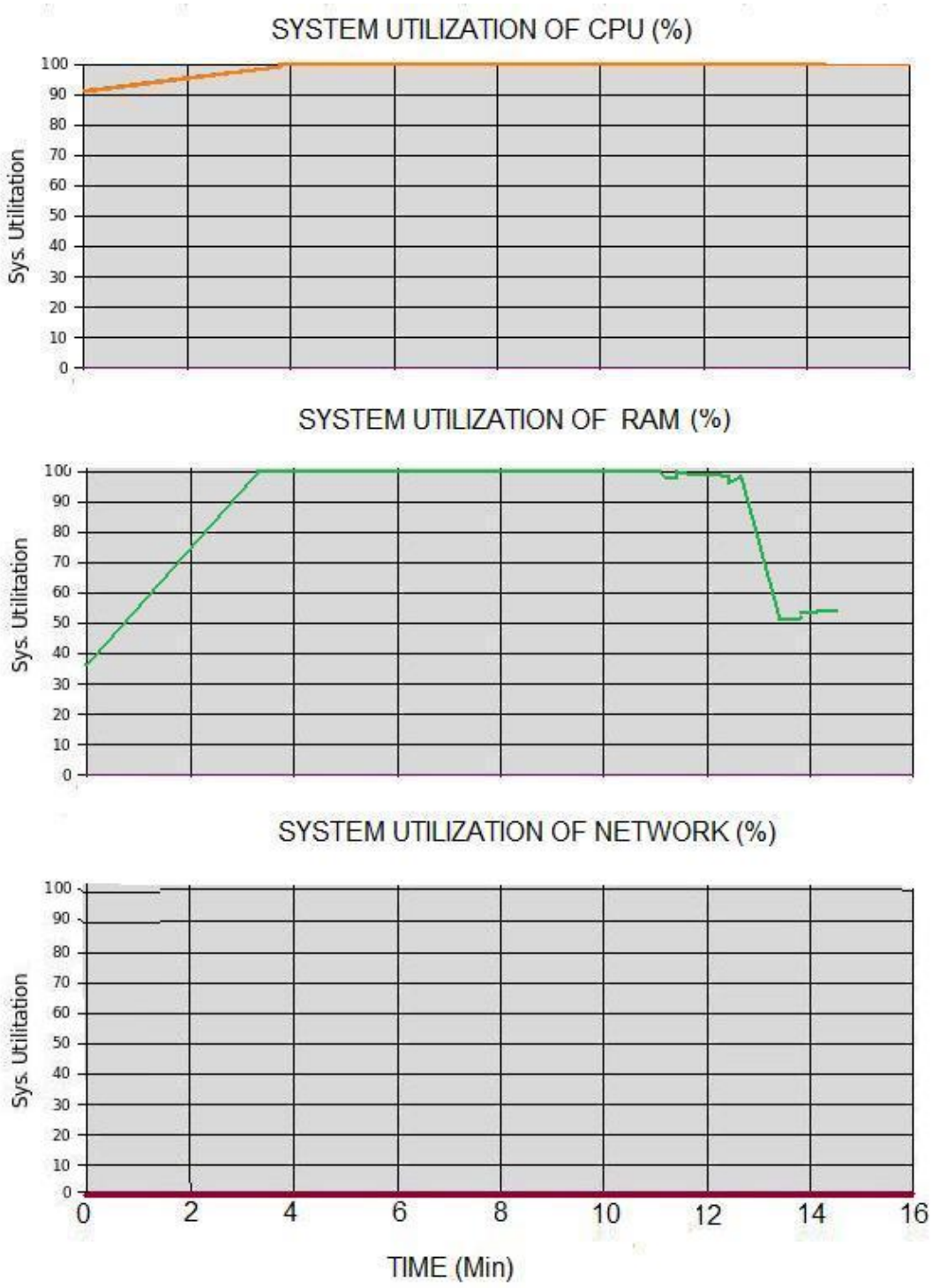


Figura 4.8: Recursos consumidos simulando la carga para 1 procesador

Una vez comprobados los resultados obtenidos para 1 nodo, procedemos a simular la misma carga de trabajo para 2, 4, 8, 16 ,32 y 64 nodos para poder estudiar los resultados obtenidos e intentar ajustar al máximo los recursos que necesitamos para lanzar las aplicaciones en el cluster real sin riesgo a sobredimensionar recursos o sobrecargar demasiado el sistema por falta de recursos. También podemos cambiar la política de planificación con el fin de intentar reducir el tiempo de ejecución total de la simulación.

En la figura 4.9 se muestra el porcentaje de utilización de CPU para una simulación con 64 procesadores utilizando una política Backfill.

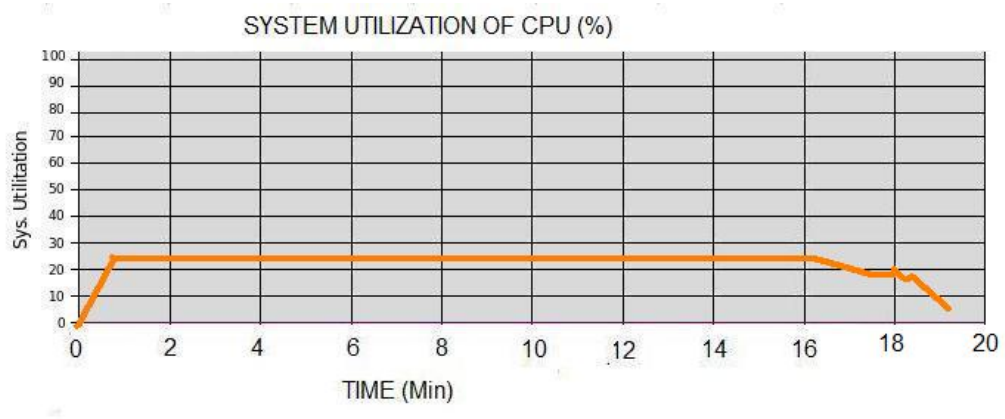


Figura 4.9: Recursos consumidos de CPU utilizando 64 procesadores y política Backfill

Las herramientas de monitorización no sólo nos permiten obtener información de los recursos consumidos durante la simulación, sino que nos ofrecen muchas más posibilidades, facilitando información sobre: el tiempo de cómputo, el tiempo de comunicaciones, el tiempo de standby, el número de mensajes enviados, etc.
La figura 4.10 nos muestra el tiempo utilizado para cómputo y el utilizado para comunicaciones para una simulación de 64 procesadores utilizando la política de planificación Backfill.

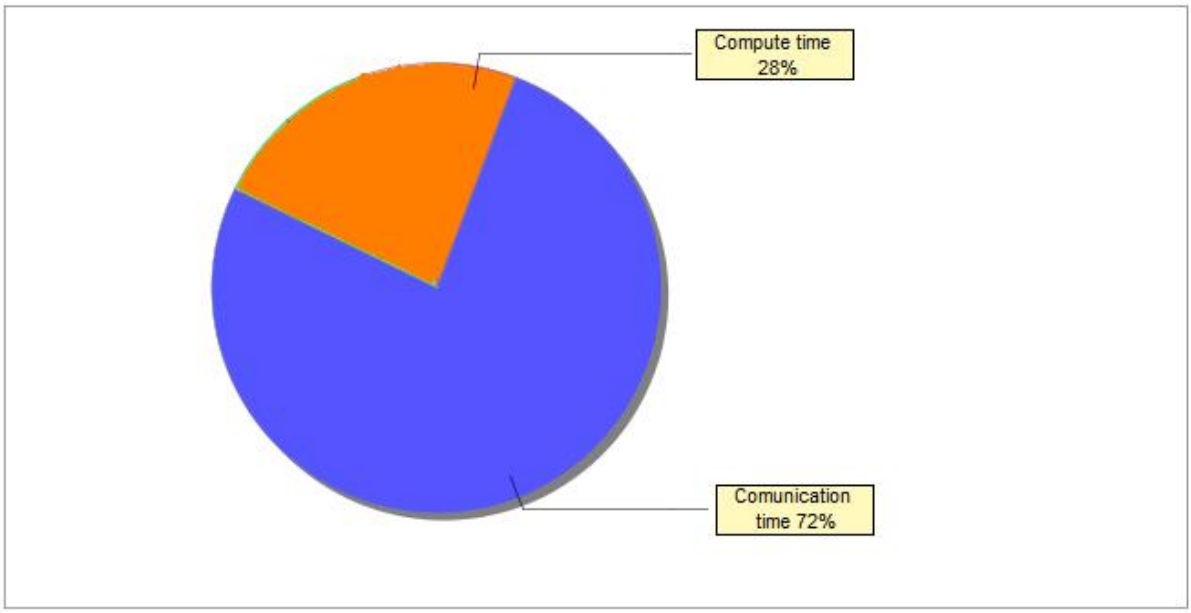


Figura 4.10: Porcentaje de tiempo empleado para cómputo y comunicaciones.

Otro aspecto a destacar es que el tiempo de la simulación no se corresponde con el tiempo de ejecución de las aplicaciones, ya que para ver lo que realmente sucede debemos separar el tiempo de standby, donde las aplicaciones están esperando en cola a poder ser ejecutadas del tiempo de cómputo (que es donde verdaderamente están las aplicaciones ejecutándose). Las herramientas de monitorización nos permiten ver el tiempo real que se ha estado ejecutando la aplicación en el cluster, y nos permite compararlo con el tiempo total.
La tabla 4.2 ilustra lo explicado previamente.

ID	Tiempo standby	Tiempo de ejecución	Tiempo total
Moab.82	00:00:00	00:00:01	00:00:01
Moab.83	00:00:02	00:00:03	00:00:05
Moab.84	00:00:04	00:00:02	00:00:06
Moab.85	00:00:06	00:00:03	00:00:09
Moab.86	00:00:10	00:00:03	00:00:13
Moab.87	00:00:13	00:00:11	00:00:23
Moab.88	00:00:24	00:00:09	00:00:33
Moab.89	00:00:32	00:00:13	00:00:45
Moab.90	00:00:48	00:00:03	00:00:51
Moab.91	00:00:52	00:13:32	00:14:24
Moab.92	00:14:26	00:00:28	00:14:52
Moab.93	00:14:54	00:00:16	00:15:10
Moab.94	00:15:09	00:00:21	00:15:30
Moab.95	00:15:32	00:00:32	00:16:04

Nota: los tiempos están redondeados a la alza para poder ver con mayor claridad lo expuesto

Tabla 4.2: Tiempos de ejecución

Como podemos observar en la tabla anterior, a partir del trabajo Moab.91 (multiplicación de matrices) el tiempo de standby es mucho mayor al tiempo de cómputo de la aplicación, con lo cual mirando el tiempo total de la ejecución de las aplicaciones observamos tiempos que no corresponden con la verdadera ejecución de la aplicación, lo que hace imprescindible tener herramientas de monitorización para poder apreciar lo que realmente está sucediendo durante la simulación.

A lo largo del capítulo se ha intentado poner de manifiesto las facilidades que nos ofrece el entorno de una forma práctica. Como conclusión, comentaremos que las interficies gráficas no ofrecen ninguna prestación extra que no podamos obtener en modo consola. Su principal cometido es facilitarnos el trabajo en este tipo de entornos.

Capítulo 5

Conclusiones

En este capítulo se detallan las conclusiones del proyecto, así como los problemas encontrados a lo largo de su desarrollo y las posibles líneas de investigación a seguir.

5.1 Variación respecto a la planificación inicial

En la figura 5.1 podemos ver la planificación que se elaboró al principio del proyecto a fin de poder cumplir todos los objetivos con éxito. Debido a los problemas que han ido surgiendo a lo largo del proyecto nos hemos visto obligados a variar el diagrama de gantt inicial para intentar cubrir los problemas que iban surgiendo y no alterar significativamente el plazo total de finalización del proyecto.

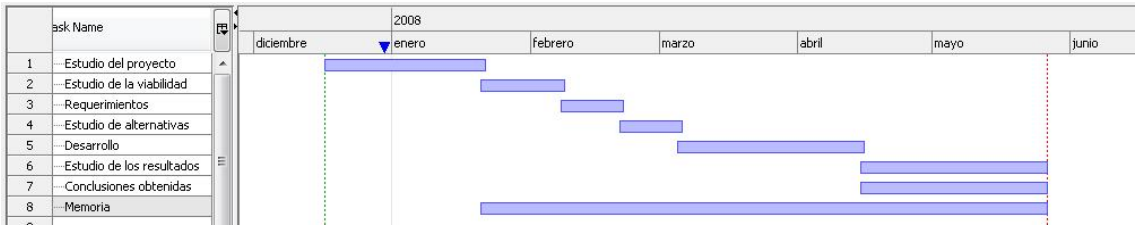


Figura 5.1: Digrama de Gantt inicial

Como se puede observar en la figura 5.1, el proyecto estaba planificado para realizarse en un tiempo aproximado de unos seis meses, a fin de poder llegar con éxito a finalizarlo a principios de junio. Las cuatro primeras etapas (Estudio del proyecto, Estudio de la viabilidad, Requerimientos y Estudio de alternativas), se llevaron a cabo con éxito en el tiempo estipulado por el diagrama de Gantt. En la etapa de desarrollo, debido a problemas para adquirir el software, ya que no era de libre distribución y las versiones de prueba no tenían habilitadas todas sus opciones, nos vimos obligados a pedir licencias a la empresa propietaria del software para poder ejecutarlo en su totalidad, con lo cual perdimos aproximadamente unas dos semanas. Este problema nos impedía poder seguir avanzando con otras tareas ya que son completamente secuenciales y siguen el modelo de cascada, es decir, no es posible iniciar una tarea si no se ha finalizado la anterior.

Una vez adquirido el software, y dispuestos a instalarlo en el cluster, vimos que después de diversos intentos comprobamos que la guía de instalación para poder instalar el software tenía algunos errores (en los anexos 1 y 2 de este trabajo podemos encontrar los manuales de instalación correctamente corregidos), con lo cual nos vimos forzados a dejar de lado el proyecto y trabajar conjuntamente con los técnicos de la empresa propietaria del software hasta poder llegar a solucionar los problemas de instalación y configuración y proceder a utilizar el software.
Todos los problemas acaecidos nos supusieron un atraso considerable en la etapa de desarrollo de aproximadamente más de un mes. Para intentar solventar dicho problema y que no afectase demasiado al tiempo de finalización del proyecto se optó por reducir en la medida de lo posible las etapas de estudio de los resultados y conclusiones obtenidas, teniendo en cuenta que son las dos etapas más importantes del proyecto y no se podían recortar excesivamente ya que eso podía comportar efectos colaterales como por ejemplo no hacer un buen estudio experimental, lo que supondría poder acabar llegando a conclusiones equivocadas.
Una de las etapas más difíciles del proyecto ha sido la etapa de estudio de los resultados debido a la complejidad del scheduler y su gran número de parámetros de configuración, lo que ha conllevó que la mayor parte de la etapa la dedicamos a entender el scheduler y sus parámetros de configuración.

La figura 5.2 ilustra como ha quedado definitivamente el diagrama de gantt una vez finalizado el proyecto.

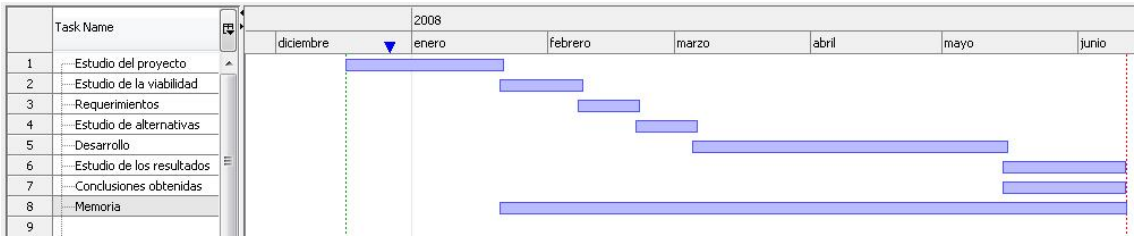


Figura 5.2: Diagrama de Gantt final

Comparando los dos diagramas podemos observar que a partir de la etapa de desarrollo el nuevo diagrama varía sensiblemente. La duración del proyecto se ha visto afectada en el término del plazo previsto inicialmente, que se ha alargado 16 días más de lo previsto.
Puesto que en la planificación inicial se tuvo en cuenta que podían suceder incidencias en las etapas del proyecto que retrasaran el plazo de finalización, se dejó un margen prudencial de tres semanas desde el plazo de finalización del proyecto a la entrega del mismo y se alargaron las etapas al máximo para poder actuar delante de cualquier incidencia.

5.2 Futuras líneas de investigación

Nuestro estudio está basado en planificación y ejecución de aplicaciones batch en clusters no dedicados, dejando a un lado las aplicaciones de tiempo real debido a la demora en el plazo previsto de finalización del proyecto y a la complejidad que suponen las aplicaciones de tiempo real.
Debido a la importancia de los sistemas de tiempo real en la industria (control de satélites, sistemas aeronáuticos, control de robots en operaciones críticas, control de procesos industriales altamente contaminantes, etc) consideramos que es un campo lo suficientemente

importante como para dedicarle todo un estudio aislado y ver las políticas de planificación y su comportamiento en clusters no dedicados.
Los sistemas de tiempo real los podemos dividir en sistemas de tiempo real estricto y sistemas de tiempo real débiles. Los sistemas de tiempo real estrictos son aquellos donde las acciones del sistema se deben producir de forma obligatoria dentro del plazo específico mientras que los sistemas de tiempo real débiles son aquellos donde la perdida puntual de algunos de sus plazos puede hacer que el sistema tenga un funcionamiento degradado durante un intervalo determinado de tiempo pero el sistema es capaz de recuperarse sin graves consecuencias. Un ejemplo de sistemas de tiempo real débil son los sistemas de video bajo demando, donde la perdida de uno o varios frames en un instante de tiempo determinado no tiene graves consecuencias para el usuario.
Los sistemas de tiempo real estricto se pueden clasificar en sistemas de planificación con prioridad fija y sistemas de planificación con prioridad dinámica. En los sistemas de planificación con prioridad fija todas las tareas de un mismo trabajo tienen la misma prioridad mientras en los sistemas con planificación dinámica las tareas tienen prioridades diferentes.
La figura 5.3 muestra una taxonomía muy general de los sistemas de tiempo real, que nos ayudará a exponer unas directrices de posibles líneas de investigación. Puesto que este tipo de sistemas está fuera del alcance de este proyecto no se ha querido profundizar en este tipo de sistemas, y solamente se han expuesto con una visión muy general para poder exponer las futuras líneas de investigación.

Figura 5.3: Sistemas de tiempo real

Como futuras líneas de investigación se propone estudiar la planificación de aplicaciones en sistemas de prioridades estáticas frente a la planificación en sistemas de prioridades dinámicas. Una vez estudiados los sistemas de tiempo real estrictos se podría ampliar el estudio y comparar la planificación en sistemas de tiempo real estricto frente a la planificación en sistema de tiempo real débiles.

5.3 Valoración personal

Después de trabajar varios meses en el estudio, planificación y desarrollo de este proyecto, puedo decir que estoy muy satisfecho con el resultado obtenido, ya que los objetivos se han cumplido satisfactoriamente.

En el momento de empezar el proyecto no tenía conocimientos de los componentes de un cluster ni conocía su funcionamiento, y a medida que he ido avanzando en el proyecto, he ido viendo su funcionamiento y la dificultad que conlleva la sintonización de sus parámetros en función de las aplicaciones; debido al gran número de ellos; que hacen que la administración de un cluster no sea algo trivial si no que se necesiten conocimientos profundos del tema y una larga experiencia de trabajo en clusters.
Una vez estudiados los clusters no dedicados y ver las posibilidades que nos ofrecen, pienso que este sistema de cómputo se podría implantar en aulas de informática, como por ejemplo las que están presentes en cualquier universidad, ya que nos permiten hacer un uso eficiente de una fuente de cómputo que está claramente desaprovechada.

Bibliografía

[1] Rajkumar Buyya. High Performance Cluster Computing Volume 1, 2004

[2] Miguel Angel Perez. Arquitecturas Paralelas, Marzo 2001

[3] M. Hanzich. A temporal and spacial scheduling system for non-dedicated cluster. PHD tesis, UAB, 2006

[4] K.Aida. Effect of job size characteristic on job scheduling performance. Job scheduling for parallel processing, 2000

[5] http://www.clusterresources.com. Página oficial de la empresa Cluster resources. Consultado en Mayo del 2008

[6] http://www.debian.org/index.es.html. Página oficial del proyecto Debian/GNU Linux. Consultado en Mayo del 2008

[7] http://www.mcs.anl.gov. Página oficial del ARGONE nacional Laboratory. Consultado en Mayo del 2008

[8] http://www.**nas**.**nas**a.gov. Página oficial de la NASA. Consultado en Mayo del 2008

Anexo 1: Instalación y configuración de Torque

En este apartado se explicará punto a punto los pasos que hay que llevar a cabo para la correcta instalación y configuración de Torque.

1- Descomprimir el software que previamente nos habremos descargado de la web oficial http://www.clusterresources.com, en el nodo servidor.

```
> tar -xzvf torqueXXX.tar.gz
```

donde XXX es la versión del software.

2-> Entramos en el directorio descomprimido y ejecutamos el script de configuración

```
> cd torqueXXX
>./configure
```

3-> Compilamos el programa

```
> make
```

4-> Instalamos el programa

```
-> make install
```

5-> Una vez tenemos instalado torque en el nodo servidor, procedemos a instalarlo en los nodos de computo, (repetimos los pasos del 1 al 4).

6-> Compilamos los archivos necesarios que se ejecutaran en los nodos de computo, para ello, vamos al directorio que hemos descomprimido anteriormente y compilamos los archivos.

```
-> cd torqueXXX
->make packages
```

debería de aparecernos por pantalla lo siguiente

```
Building ./torque-package-clients-linux-i686.sh ...
Building ./torque-package-mom-linux-i686.sh ...
Building ./torque-package-server-linux-i686.sh ...
Building ./torque-package-gui-linux-i686.sh ...
Building ./torque-package-devel-linux-i686.sh ...
Done.
```

7-> Una vez tenemos construidos los paquetes de los nodos servidores, procedemos a su instalación, en los nodos de computo.

```
-> /torque-package-mom-linux-i686.sh --install
->torque-package-clients-linux-i686.sh --install
```

8-> Una vez tenemos instalado torque en el nodo servidor y en los nodos de cómputo, procedemos a configurar el nodo servidor.
En la guía de instalación de torque nos sugiere que hagamos lo siguiente "pbs_server -t create" con lo que nos creará una base de datos vacía que habremos de rellenar manualmente. Nosotros utilizaremos un script, que aunque no está en la guía de instalación nos hará ese paso automaticamente y nios rellenará la base de datos automaticamente

```
-> torque.setup <USER>
```

donde user no debe de ser superusuario, ya que sino tendremos conflictos al integrarlo con moab

9-> Una vez tenemos el nodo servidor bien configurado procedemos a ejecutar añadir los nodos de computo que queramos que formen parte del cluster.
En la guía de instalación se nos recomienda editar el archivo $TORQUE_HOME/server_priv/nodes y añadir manualmente el nombre de cada nodo. Nosotros omitiremos ese paso debido a que al añadirlo manualmente el archivo se daña y no entiende el nombre de los nodos.
Para añadir los nodos utilizaremos la siguiente sentencia

```
> qmgr -c "create node nombre_del_nodo_a_añadir"
```

en caso de habernos equivocado en el nombre del nodo y querer eliminarlo, tampoco lo haremos editando el archivo sino que lo haremos de la siguiente manera.

```
> qmgr -c "delete node nombre_del_nodo_a_eliminar"
```

para asegurarnos que los nombres del nodo son correctos, ejecutaremos la siguiente sentencia

```
more $TORQUE_HOME/server_priv/nodes
```

10-> Una vez tenemos configurado todo el cluster procedemos a testear la instalación.

```
# Paramos el servidor
> qterm

# Iniciamos el demonio servidor
> pbs_server

# Iniciamos los demonios en los nodos de cómputo
> pbs_mon

# como todavía no tenemos torque integrado con moab
# iniciamos el planificador de moab por defecto
> pbs_sched
```

```
# comprobamos que las colas están bien configuradas
> qstat -q

# Comprobamos la configuración adicional del servidor
>  qmgr -c 'p s'

# comprobamos que los nodos son correctos
> pbsnodes -a

# Enviamos un trabajo básico al nodo para comprobar si lo ejecuta
> echo "sleep 30" | qsub

# comprobamos el estado del trabajo
> qstat
```

Anexo 2: Instalación y integración de Moab con Torque

En este apartado se explicará punto a punto los pasos que hay que llevar a cabo para la correcta instalación de Moab y posteriormente su integración con Torque

1- Descomprimir el software que previamente nos habremos descargado de la web oficial http://www.clusterresources.com, en el nodo servidor.

```
> untar MoabXXX
```

donde XXX es la versión del software.

2-> Entramos en el directorio descomprimido y ejecutamos el script de configuración

```
> cd torqueXXX
>./configure
```

3-> Compilamos el programa

```
> make
```

4-> Instalamos el programa

```
> make install
```

5-> ejecutamos la siguiente sentencia para que moab reciba información en tiempo real de Torque

```
>qmgr 'set server scheduling=True'
```

6-> Una vez echo esto tenemos que copnfigurar el archivo moab.cfg, dicho archivo lo podemos encontrar dentro del directorio de moab /opt/moab.

```
> cd /opt/moab
> joe moab.cfg
```

7-> Debido que la configuración de este archivo para la correcta integración con Moab no está documentado correctamente en la guíá de instalación, podreceré a poner el archivo tal y como tiene que quedar para su correcto funcionamiento.

```
SCHEDCFG[Moab]      SERVER=nombre_host_servidor:42559
SCHEDCFG[Moab]      MODE=NORMAL

ADMINCFG[1]         USERS=root
TOOLSDIR            /usr/local/tools
LOGLEVEL            3

RMCFG[base]         TYPE=PBS
RMCFG[base]         SUBMITCMD=/usr/local/bin/qsub



#Simulation  Setting
SIMRESOURCETRACEFILE resource.trace
SIMWORKLOADTRACEFILE workload.trace
SIMSTOPOPERATION    0
SIMTIMEPOLICY       REAL
JOBSUBMISSIONPOLICY  CONSTANTJOBDEPTH
SIMTIMERATIO        0
```

nota: En la primera linea hay que poner el hostname de el host servidor

8-> En el paso anterior hemos configurado Moab para trabajar con Torque en modo normal, para trabajar en modo simulación y puesto que ya hemos puesto en el paso anterior las directivas de simulación bien configuradas lo único que habría que hacer son los siguientes cambios

```
#Comentamos la siguiente linea
#RMCFG[base]        TYPE=PBS

#cambiamos en la segunda linea el parametro NORMAL por SIMULATION,
#con lo cual nos quedaría lo siguiente

SCHEDCFG[Moab]      MODE=SIMULATION
```

9-> Una vez tenemos integrado Moab con Torque, procedemos a probar su correcto funcionamiento.
Para comprobar su funcionamiento consideramos que estamos trabajando en modo normal(paso 7).

```
# inciamos moab
> moab
```

```
# inciamos el servidor de Torque
> pbs_serv

# inciamos algún nodo de cómputo para poder ejecutar algún trabajo
> pbs_mom

# enviamos algún trabajo al cluster
> echo “sleep 30” |msub

#comprobamos el estado de la cola

>showq
```

nota: Para ejecutar los servidores de torque y moab se tiene que hacer en modo superusuario, mientras que para enviar trabajos hay que hacerlo con el usuario con el cual configuramos Torque (Anexo 1 paso 8)

Anexo 3: Instalación de MPICH2

1-> Descargamos el software de su página oficial, http://www.mcs.anl.gov/index.php.

2-> Descomprimimos el archivo

```
> tar xfz mpich2.tar.gz
```

3-> Creamos un directorio de instalación

```
> mkdir /home/you/mpich2-install
```

4-> Creamos un directorio de compilación y copiamos el archivo descomprimido

```
> mkdir /tmp/you/mpich2-1.0.7
> cp XXX  /tmp/you/mpich2-1.0.7
```

donde XXX es la ruta del archivo descomprimido

5-> Ejecutamos el script de configuración indicándole el directorio de instalación.

```
> cd /tmp/you/mpich2-1.0.7

> ./configure -prefix=/home/you/mpich2-install
```

6-> Construimos el archivo

```
> make
```

7->Instalamos el archivo

```
> make install
```

8-> Exportamos el subdirectorio bin a nuestro path para poder usar los comandos desde cualquier
lugar

```
> export PATH=/home/you/mpich2-install/bin:$PATH
```

9-> Comprobamos que la exportación ha sido correcta

```
> which mpd

> which mpicc
```

```
> which mpiexec

> which mpirun
```

nos debería de aparecer, lo siguiente para todas las instrucciones

```
$ /bin/mpd
$ /bin/mpicc

$ /bin/mpiexec

$ /bin/mpirun
```

10-> Una vez tenemos la instalación básica instalada procederemos a configurar el anillo de computadoras que formaran parte de la comunicación.
Primeramente y como método de seguridad, crearemos un archivo con una contraseña, en nuestro directorio.
En esta tarde el manual de la web oficial esta equivocado, ya que no edita corrrectamente el archivo de contraseñas, es recomendable seguir esté punto tal y como está aquí.

```
> cd $HOME

> touch .mpd.conf

> chmod 600 .mpd.conf
> joe
.mpd.conf
```

en el archivo poner la siguiente linea de texto cambiando la contraseña

```
> secretword=contraseña
```

11-> Comprobamos que la instalación es correcta y el host responde bien

```
# ejecutamos el demonio de MPICH
mpd &


#Sentencia que nos devolvera al nombre de nuestro host si todo esta correcto
mpdtrace


#Paramos el demonio de MPICH
mpdallexit
```

12-> Una vez tenemos configurado el anillo para una máquina, lo configuramos para las demás, teniendo en cuenta que tienen que estar dentro de un mismo dominio para su correcto funcionamiento.
Para ello creamos un archivo en nuestro directorio donde ponemos el hostname de los host que participarán en la comunicación.

```
> cd $HOME

> touch mpd.host
> joe mpd.host
```

13-> Una vez está todo bien instalado y configurado ponemos en marcha el entorno

```
> mpdboot -v -n num_proc -f fichero_maquinas
```

14-> Compilamos el programa

```
> mpicc  -o programa programa.c
```

15 -> Ejecutamos el programa compilado

```
> mpiexec -n xx programa #donde XX es el numero de procesos.
```

Anexo 4: Instrucciones básicas de Torque/Moab

Comando	Descripción
msub	Envia trabajos al cluster
showq	Muestra el estado de la cola de ejecución
canceljob	Cancela un trabajo determinado de la cola de ejecución
runjob	Fuerza un trabajo a ser ejecutado inmediatamente
checkjob	Muestra detalles del estado de un trabajo concreto
checknode	Muestra detalles del estado de un nodo en concreto
mshow	Muestra mensajes sobre el estado del sistema y las colas
mjobctl	Controla y modifica trabajos
mnodectl	Controla y modifica nodos
mdiag	Muestra el estado de los recursos, workload y sheduling
mschedctl	Modifica el estado y el comportamiento del scheduler
resetstats	Resetea las estadísticas del scheduler
showstart	Nos dice cuando un trabajo podrá empezar a ejecutarse
showstate	Nuestra el estado actual de los recursos
showstats	Muestra las estadísticas de uso
showstats – f	Muestra varias tablas con información y características del scheduler

Anexo 5: Comandos de especificaciones TORQUE/PBS

Comando	Significado	Unidades
Arch	Permite seleccionar la arquitectura del sistema	string
ncpus	Numero de procesadores requeridos para ejecutar un trabajo	entero
mem	Memoria total requerida para ejecutar un trabajo	size
nodes	Numero y tipo de nodos necesarios para ejecutar el trabajo	node_spec
File	Espacio máximo de disco para un archivo simple creado por el trabajos	size
cput	Cantidad total de tiempo de cpu requerido por todos los procesos en el trabajo	time
walltime	Intervalo máximo de tiempo real que un trabajo puede estar en estad o running	time
pmem	Máxima cantidad de memoria física (workingset) usada por cualquier proceso del trabajo	size
pcput	Máxima cantidad de tiempo de CPU usado por cualquier proceso del trabajo	time
pvmem	Máxima cantidad de memoria virtual usada por cualquier proceso del trabajo	size
software	Permite al usuario especificar un software requerido por el trabajo	string
vmen	Cantidad máxima de memoria virtual usada por todos los procesos concurrentes del trabajo	size
-a fecha_hora	Aplaza la ejecución de un trabajo un determinado instante de tiempo	number
-c intervalo	Especifica el intervalo de tiempo entre cada checkpoint	number
-e path	Redirecciona la salida del fichero de error	string
-I	El trabao será ejecutado interactivamente	--
-j	Fusiona los archivos de salida y de error en uno solo	string
-k keep	Retiene los ficheros de salida	string
-l nodespec	Especificacion del nodo	string
-M lista usuarios	Usuarios a quien es enviado el mail de confirmación una vez se ha ejecutado el trabajo	string
-m	Envia un mail de notificación a los usuarios de la lista una vez el trabajo se ha ejecutado	--
-N nombre	Especifica el nombre del trabajo	string
-o path	redirecciona los archivos de salida	string
-p prioridad	Establece una prioridad al trabajo	entero

-q destinación	Especifica la cola de ejecución donde será ejecutado el trabajo	string
-r valor(Y o N)	Especifica si un trabajo es re ejecutable	caracter
-S path	Especifica el tipo de shell utilizada	path
-u lista_usuarios	Especifica el usuario o usuarios propietarios del trabajo	string
-V	Exporta las variables de entorno	--
-v lista_variables	Extiende el conjunto de variables de entorno	string
-W depend= lista	Especifica el grupo o grupos propietarios del trabajo	string

Anexo 6: Parámetros configuración Moab

Parametro	Descripción
ACCOUNTCFG[x]	Especifica unos atributos concretos a una determinada ID
ACCOUNTINGINTERFACEURL	Especifica la interface para importar información a un sistema de tiempo real
ACCOUNTWEIGHT	Especifica la prioridad que será aplicada
ADMINCFG[X]	Especifica los niveles de administración de cada usuario
ALLOWROOTJOBS	Especifica si un administrador tiene permitido lanzar trabajos
BACKFILLDEPTH	Especifica el numero de trabajos que serán evaluados por la política Backfill. El algoritmo backfill evaluará el numero de trabajos indicados basándose en su prioridad.
BACKFILLMETRIC	Criterio para poder decidir que trabajo es el mejor según la política backfill
BACKFILLPOLICY	Especifica que variante del algoritmo de backfill será aplicado
BFPRIORITYPOLICY	Especifica el atributo a utilizar a la hora de dar prioridad a los trabajos con el algoritmo backfill
CHECKPOINTFILE	Especifica el archivo de checkpoint
CHECKPOINTINTERVAL	Especifica el intervalo de cada checkpoint
CLIENTMAXPRIMARYRETRY	Especifica el número de veces que el cliente intentará volver a conectarse con el servidor si este no está disponible la primera vez
CLIENTMAXPRIMARYRETRYTIMEOUT	Especifica cuanto tiempo espera un cliente para intentar volver a conectar con el servidor.
CLIENTTIMEOUT	Especifica el número de tiempo que el cliente esperará una contestación del servidor a un comando proporcionado.
DEADLINEPOLICY	Especifica que hacer cuando se produce un deadline
DEADLINEPOLICY	Especifica el gestor de recursos por defecto para interpretar los comandos y los scripts de especificaciones
DEFAULTSUBMITPARTITION	Especifica la partición por defecto donde ira el trabajo.
DEFERCOUNT	Especifica el número de veces que un trabajo puede ser aplazado hasta cambiar al estado hold
DEFERTIME	Especifica la cantidad de tiempo que un trabajo estará en estado aplazado(HOLD) antes de ser devuelto al estado IDLE.
DIRECTORYSERVER	Especifica la dirección y el puerto del servidor
DISABLESLAVEJOBSUBMIT	Impide que los usuarios puedan lanzar trabajos desde cualquier nodo de cómputo
DISPLAYFLAGS	Especifica el flag que moab utilizará para mostrar información de la cola de ejecución

EMULATIONMODE	Creará un entorno de ejecución similar al gestor de recursos que le pasemos como argumento
ENABLENEGJOBPRIORITY	Permite al scheduler variar los trabajos en un rango de entre -INFINITY y MMAX_PRIO
ENABLEPOSUSERPRIORITY	Permite especificar a los usuarios prioridades a los trabajos en un rango de -1023 a +1024
ENABLESTARTESTIMATESTATS	Moab cogerá todas las estadísticas de los trabajos ejecutados ejecutados para intentar mejorar futuras estimaciones
ENFORCEACCOUNTACCESS	Especifica si Moab debe de cumplir las restricciones de acceso sin una asignación de recursos.
FEATURENODETYPEHEADER	Especifica la cabecera del nodo utilizado para obtener el tipo y características del nodo
FEATUREPARTITIONHEADER	Especifica la cabecera utilizada para especificar la partición del nodo
FEEDBACKPROGRAM	Especifica el nombre del programa que se ejecutará al acabar el trabajo
FORCERSVSUBTYPE	Especifica que los recursos reservados del administrador deben de tener un subtipo asociado a ellos
GRESCFG[X]	Especifica asociaciones de recursos genéricos a un grupo de recursos
GRESTOJOBATTRMAP	Lista de recursos genéricos
ENABLEHIGHTHROUGHPUT	Configura moab para que pueda aceptar trabajos de manera que reduzca al mínimo su tiempo de procesado
GROUPCFG[X]	Especifica unos atributos especificos para un grupo determinado
GROUPWEIGHT	Asigna una prioridad a un grupo específico
IDCFG[X]	Interfaz que permite al administrador compartir información con un servicio externo
IGNORECLASSES	Ignora los trabajos providentes de las clases de trabajos
IGNOREJOBS	Ignora todos los trabajos excepto la lista que recibe como argumento
IGNORENODES	Moab ignora todos los nodos excepto la lista que recibe como argumento
IGNOREUSERS	Moab ignora todos los usuarios de la lista que recibe como argumento
JOBACTIONONNODEFAILURE	Especifica la acción a tomar si un nodo con un trabajo activo falla
JOBAGGREGATIONTIME	Especifica la cantidad de tiempo que el scheduler debería esperar desde que recibe un trabajo hasta que procesa dicho trabajo
JOBCFG[X]	Especifica atributos concretos para un trabajo específico

JOBCPURGETIME	Especifica la cantidad de tiempo que moab guarda las estadísticas de los trabajos ejecutados
JOBMAXHOLDTIME	Especifica la cantidad de tiempo que podemos aplazar un trabajo antes de ser cancelado
JOBMAXNODECOUNT	Determina la cantidad máxima de nodos que se pueden utilizar para ejecutar un trabajo
JOBMAXPREEMPTPERITERATIO	Número máximo de trabajos permitidos por iteración
JOBPRIOACCRUALPOLICY	Especifica como los aspectos dinámicos del trabajo será ajustados
JOBREJECTPOLICY	Especifica la acción a tomar cuando el scheduler determina que un trabajo nunca se puede ejecutar
LOGDIR	Especifica donde se guardarás los ficheros de log
LOGFACILITY	Especifica que tipos de eventos se almacenarán en el log
LOGFILEMAXSIZE	Tamaño máximo del archivo de log
LOGFILEROLLDEPTH	Número máximo de ficheros de log permitidos
MAXJOB	Número máximo de trabajos que pueden ser ejecutados simultaneamente por el scheduler
MAXRSVPERNODE	Especifica el máximo número de reservas que puede tener un nodo
MEMWEIGHT	Especifica el coeficiente por el cual será multiplicado la cantidad de memoria que necesite un trabajo
MINADMINSTIME	Especifica el tiempo mínimo que un trabajo será suspendido, en caso de que sea el administrador el que lo haya suspendido
NODEACCESSPOLICY	Especifica como los recursos del nodo serán compartidos por varias tareas
NODEALLOCATIONPOLICY	Especifica como moab debería repartir los recursos disponibles a los trabajos
NODEAVAILABILITYPOLICY	Especifica como moab debería evaluar la disponibilidad de un nodo
NODECFG[X]	Especifica unos atributos en concreto para el nodo indicado
NODEMAXNODE	Indica la cantidad máxima de carga de cpu que puede usar un trabajo
NODEPOLLFREQUENCY	Especifica las iteraciones que se tienen que realizar desde que se ha iniciado el scheduler hasta que se pueden hacer consultas al mismo
PARALLOCATIONPOLICY	Especifica un criterio para asignar recursos cuando más de una partición estén libres
PARACFG[X]	Especifica los atributos, políticas y limitaciones para una partición
QOSCFG[X]	Especifica unos atributos específicos a la política QoS

QOSREJECTPOLICY	Especifica la acción a tomar cuando un trabajo no puede acceder a una petición QoS
QOSWEIGHT	Especifica el peso de la prioridad que se ha aplicado a cada trabajo
REMAPCLASS	Especifica el número de procesadores que se pueden asignar a cada clase/cola
RMCFG[X]	Especifica que gestor de recursos utiliza el scheduler
RSVCTLPOLICY	Especifica quien puede reservar recursos aparte del administrador
SIMULATIONSHUTDOWN	La simulación se termina cuando la cola queda vacía
SIMCPUSCALINGPERCENT	Especifica si incrementar o decrementar el reloj del runtime de cada trabajo
SIMIGNOREJOBFLAGS	Especifica que atributos o criterios ignorará el simulador del fichero de trazas
SIMINITIALQUEUEDEPTH	Especifica cuantos trabajos pueden estar en la cola de ejecución con estado IDLE durante la simulación
SIMJOBSUBMISSIONPOLICY	Especifica como el simulador subministra nuevos trabajos a la cola durante la simulación
SIMNODECOUNT	Especifica el máximo número de nodos que pueden utilizarse para la simulación
SIMPURGEBLOCKEDJOBS	Especifica si moab debería eliminar de la simulación trabajos que no se van a poder ejecutar nunca
SIMRESOURCETRACEFILE	Especifica el fichero de recursos de la simulación
SIMSTARTTIME	Especifica en que instante de tiempo comenzará la simulación
SIMSTOPOPERATION	Especifica en que iteracion moab deberá parar para esperar comandos de usuario
SIMTIMEPOLICY	Especifica el paso de tiempo de forma real durante la simulación
SIMTIMERATIO	Determina el speedup del walltime
SIMWORKLOADTRACEFILE	Especifica el fichero de trazas de la simulación
VCPROFILE	Determina los atributos del cluster virtual

RESUMEN

El presente trabajo expone los entornos de desarrollo de aplicaciones paralelas en sistemas distribuidos. Se ha dedicado especial atención a los entornos gráficos, como plataforma de acceso al cluster. Dada la cantidad de alternativas a seleccionar en este tipo de entornos, como puede ser la parametrización de las aplicaciones, especificación de la arquitectura a utilizar de los nodos, caracterización de la carga, políticas de planificación, etc, hace que este trabajo pueda servir de punto inicial de partida a usuarios que deseen iniciarse en este tipo de entornos

RESUM

El present treball exposa els entorns de desenvolupament d'aplicacions paral·leles en entorns distribuïts. Hem dedicat especial atenció als entorns gràfics, com la plataforma d'accés al cluster. Atesa la quantitat d'alternatives seleccionables en aquest tipus d'entorns, com pot ser la parametrització de les aplicacions, especificació de l'arquitectura a utilitzar dels nodes, caracterització de la càrrega, polítiques de planificació, etc, fa que aquest treball pugui servir de punt de partida inicial per a usuaris que desitgin iniciar-se en aquest tipus d'entorns.

SUMMARY

This project explain the development environments parallel applications in distributed systems. We have devoted special attention to graphical environments as a platform for access to cluster. Given the number of alternatives to choose in such environments such as parameterization of applications, specifying the use of architecture of nodes, characterization of loading, planning policies, etc., We think that this project may serve as a point starting for initial users wishing to take up this kind of environments.

Javier Panadero Martínez