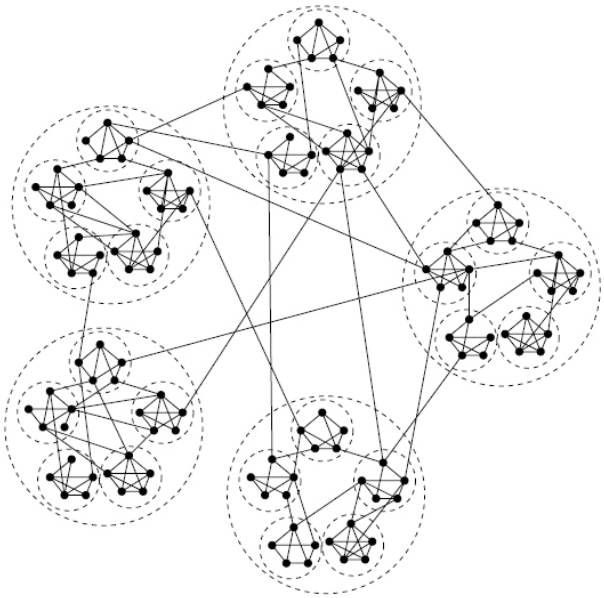
Trabajo Final de Grado:

Configuración y análisis de las prestaciones de un sistema multicomputador para la ejecución de algoritmos complejos



Autor: **Jose E. Torres Penalva**
Director: **David Cuesta Frau**

Febrero 2017

Grado en Ingeniería Informática

UNIVERSITAT POLITÈCNICA DE VALÈNCIA
CAMPUS D'ALCOI

Agradecimientos

A mi familia por darme la oportunidad.
A las personas que brindaron su apoyo en el momento exacto.
A todos los profesores que fueron inspiración, motivación y una referencia.

Resumen

Actualmente, la utilización de supercomputadores está ampliamente extendida en los ámbitos de la ingenieria y la investigación, como un pilar más para afrontar los complejos procesos de cálculo y simulación. Debido a esta necesidad, las arquitecturas de computadores evolucionaron para proporcionar un incremento en el rendimiento cuando la propia evolución de la arquitectura de procesadores empezó a encontrar limitaciones de potencia a nivel individual.

En esta línea, el presente proyecto trata de explorar las posibilidades que un sistema multicomputador obtenido a partir de computadores antiguos y en desuso, puede brindar a desarrolladores e investigadores. Reaprovechando sistemas obsoletos, y a un coste muy bajo, es posible obtener sistemas multicomputadores de tipo clúster operando bajo una distribución de sistema operativo GNU/Línux para realizar cálculos complejos de forma eficiente, como se ilustrará en este proyecto.

En concreto, los objetivos del proyecto están orientados a disponer de un laboratorio de pruebas, en el contexto docente y de investigación, donde se ilustren las distintas fases de implementación del sistema: distribución y topología de máquinas, proceso de instalación, pautas en la configuración, utilización de herramientas relacionadas en el ámbito de cálculo distribuído con la ejecución de programas en MPI, y posterior análisis del rendimiento.

keywords: multicomputador, cluster, supercomputación, hpc, linux, administración, monitorización, paralelismo, MPI, profiling, performance, eficiencia

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

En el siguiente documento se incluye la documentación del proyecto realizado como Trabajo Final de Grado con el título, Configuración y análisis de las prestaciones de un sistema multi-computador para la ejecución de algoritmos complejos, del Grado en Ingeniería Informática de la (UPV) Escola Politécnica Superior de Alcoi.

El proyecto se basa en el análisis y configuración de un sistema cluster utilizando unos pocos ordenadores, además de una revisión de algunas de las herramientas más significativas y las bondades de este tipo de sistema para su explotación en un entorno de investigación.

En el segundo capítulo se realizará un análisis superficial del diseño del cluster, basándose en el hardware disponible y el sistema operativo elegido.

En el tercer capítulo se describirá la configuración del sistema, tanto a nivel hardware como software.

En el cuarto capítulo se realizarán test de rendimiento basados en MPI, para analizar algunos de los efectos de la utilización de este tipo de sistema así como resultados y comparativas.

En el quinto capítulo se introducirán algunas herramientas de análisis de métricas sobre la ejecución de programas en este tipo de sistema, útiles para diagnosticar perdidas de eficiencia en ejecución.

En el sexto capítulo se estimará el coste eléctrico de la utilización del sistema en diferentes rangos de tiempo.

En el séptimo capítulo se incluirán algunas conclusiones del trabajo realizado.

1.1 Motivación

Para suplir la demanda de más potencia de cómputo destinada para afrontar diferentes problemas en ámbitos de investigación e ingeniería, se desarrollaron computadores paralelos para ofrecer una mayor capacidad de cálculo sumando los recursos de múltiples unidades de procesamiento actuando como todo un conjunto.

Existieron razones para desarrollar sistemas de este tipo, destacando las limitaciones de los materiales utilizados para incorporar más integración y poder conseguir frecuencias más altas en sistemas mono-procesador. Es por ello que, inicialmente se optó por la utilización de múltiples procesadores trabajando conjuntamente, y así sobrepasar la barrera de potencia en ejecución secuencial.

Aunque la conexión de múltiples procesadores permitió rápidamente incrementar el rendimiento debido a las mejoras en el número de procesadores conectados y canales de interconexión más veloces, este método se basaba en soluciones con un alto tiempo de desarrollo y alto coste para mejorar la integración de procesadores y conseguir un mejor rendimiento, sumando además las limitaciones en la escalabilidad al tratar de conectar numerosos procesadores entre ellos y su interconexión.

Existían además una serie de necesidades que se tenían que satisfacer, donde se podrian destacar las siguientes.

- **Mayor potencia de cálculo:** la necesidad de configuraciones de sistemas que incluyeran múltiples procesadores trabajando conjuntamente para obtener una mayor capacidad de cálculo y rendimiento al sumar todas las potencias.
- **Ratio coste/rendimiento:** la necesidad de sistemas de bajo coste tanto a nivel de adquisición como de consumo eléctrico.
- **Naturaleza de problemas:** la necesidad de resolver problemas basados en estructuras abordables desde un planteamiento paralelo.
- **Escalabilidad:** la necesidad de sistemas que permitieran afrontar un crecimiento en las cargas de trabajo sin que ello afectara de manera significativa al rendimiento.

La orientación de la supercomputación y el paralelismo en los primeros sistemas multiprocesador, no tardó en encontrar problemáticas para cumplir algunas de las necesidades anteriores.

Por ello, para satisfacer los requisitos anteriores y gracias al desarrollo de tecnologías de redes de interconexión más rápidas y la disponibilidad de computadores individuales de notable rendimiento con un menor coste, se decantó por la interconexión de este tipo de máquinas entre sí, conformando lo que se conoce como arquitectura multicomputador o multiprocesador de paso de mensajes.

Dentro de esta arquitectura, denominada usualmente cluster, su arquitectura está conformada por múltiples ordenadores de características hardware similares, interconectados entre sí actuando como una sola máquina. El desarrollo del proyecto se basa en la implantación de un sistema reducido basado en esta arquitectura y analizar algunas de sus utilidades tanto a nivel de administración como de potencia de cálculo.

1.2 Paralelismo

Es interesante remarcar, que cualquier aplicación desplegada en un sistema con múltiples unidades de procesamiento, para beneficiarse de la división de procesos, debe de tener en cuenta los distintos grados de paralelismo existentes para afrontar el desarrollo de una aplicación de manera eficiente y acorde al programa planteado. Existen diferentes tipos de paralelismo:

- **Paralelismo de control** basado en la capacidad de realizar diferentes operaciones de forma concurrente y de manera independiente. Implica el control de dependencias por los recursos sin perjudicar al rendimiento de la solución.
- **Paralelismo de datos** basado en la capacidad de realizar una instrucción sobre datos estructurados compuestos. Esta explotación presenta la dificultad de adecuar la capacidad de cómputo de la máquina al volumen de datos para garantizar un rendimiento apropiado.
- **Paralelismo de flujo** basado en la naturaleza de algunos problemas donde se deben realizar diferentes operaciones sobre el mismo flujo de datos. Esto plantea que la ejecución de una sección del programa tiene que desembocar en otra sección ubicada en otra unidad de proceso, bifurcándose para poder continuar con la ejecución del problema y la sincronización que ello conlleva.

1.3 Introducción a arquitecturas

Durante las ultimas décadas, las arquitecturas de computadores han evolucionado rápidamente a través de distintas configuraciones máquina entre componentes tanto en número como en distribución, partiendo desde la clásica distribución Von Neumann a modelos más complejos y desarrollados de sistemas como procesadores vectoriales, matriciales, supercomputadores, etc.

Esta evolución, acorde en parte con la Ley de Moore que determina la evolución continua del rendimiento, brindo la posibilidad de explotar estas nuevas configuraciones para poder obtener un nivel más alto de rendimiento en el tiempo de ejecución para algunas aplicaciones, dando la posibilidad así de resolver algunos problemas de forma más rápida y eficiente.

A raíz de esos cambios hardware, los modelos de programación adoptaron una visión más directa de la explotación de paralelismo en estas máquinas para poder obtener los máximos valores de rendimiento, siendo este el principal mecanismo para sacar partido a los nuevos sistemas además de abarcar otros problemas ante la rápida evolución de las arquitecturas.

Basándose en la Clasificación de Flynn es interesante diferenciar diferenciar las siguientes categorías:

- **SISD (Single Instruction/Single Data):** Dentro de este grupo entran los computadores donde existe solo un flujo de instrucciones y un flujo de datos. Conocido como escalar. Totalmente secuencial.
- **MISD (Multiple Instruction/Single Data):** Múltiples instrucciones sobre un flujo individual de datos, considerada impracticable, aunque otras interpretaciones pueden incluir algunas configuraciones dentro de este grupo.
- **SIMD (Single Instruction/Multiple Data):** un flujo de instrucciones sobre múltiples flujos de datos. Basada en varias unidades de procesamiento gobernadas por una unidad de control, ejecutando una instrucción sobre múltiples datos. Procesadores vectoriales.

- **MIMD (Multiple Instrucion/Multiple Data):** Múltiples flujos de instrucciones sobre múltiples flujos de datos. Usualmente compuestas por varias unidades de proceso. Ofrecen un alto nivel de paralelismo.

Además de esas clasificaciones principales, la evolución de algunas arquitecturas desarrolla clasificaciones más detalladas en función del tipo de funcionamiento que presentan. Relacionadas directamente con la temática de este documento basándose en la clasificación MIMD, se podrían destacar:

- **Multiprocesadores:** Este tipo de arquitectura está compuesta por varias unidades de proceso interconectadas entre sí por un bus donde usualmente se comparte la misma memoria (memoria compartida).
- **Multicomputadores:** Esta arquitectura es un conjunto de diferentes computadores (nodos) conectados entre sí donde cada uno dispone de su propia unidad de proceso y memoria local. La suma de todas estas memorias interconectadas resultan en la memoria total del sistema, donde el acceso a segmentos de memoria no locales de cada procesador se realiza usando tecnología de paso de mensajes a través de la red de interconexión. Este tipo de arquitectura es usualmente conocida como sistema de memoria distribuida, donde en algunos casos, se puede construir un direccionamiento virtual de la memoria.

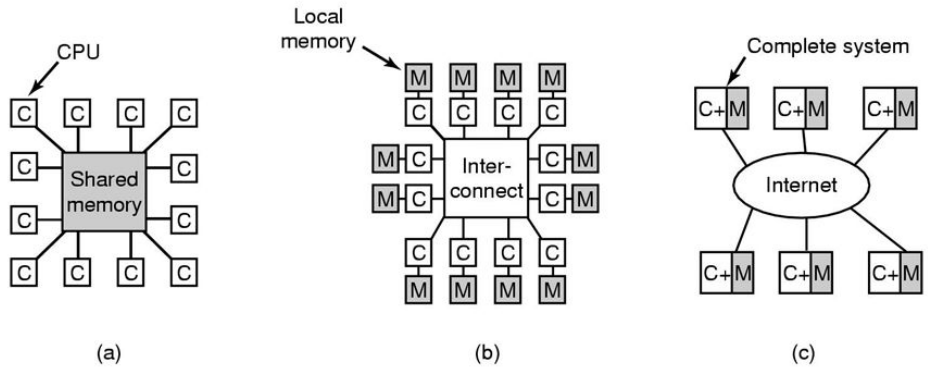


Figura 1.1: Representacion gráfica de diferentes arquitecturas multicomputador

En la figura 1.1 podemos visualizar diferentes arquitecturas multicomputador.

- **(a)** Arquitectura **multiprocesador** de memoria compartida
- **(b)** Arquitectura **multicomputador** de memoria distribuida
- **(c)** Arquitectura distribuida de **interconexión de clusters**

1.4 Multicomputadores

Durante en el desarrollo de arquitecturas multiprocesador, se encontraron algunos inconvenientes considerados en los siguientes puntos:

- Se necesitaban técnicas de sincronización entre procesadores para controlar el acceso a los recursos o datos del sistema y resolver así algunas de las dependencias.
- El acceso concurrente a través del bus realizado por múltiples procesadores puede provocar contención en el acceso a memoria y reducir significativamente la velocidad del sistema.
- Los sistemas multiprocesador no son fácilmente ampliables cuando se pretende acomodar un gran número de procesadores trabajando conjuntamente.

Como una solución a la escalabilidad de máquinas multiprocesador y evitar los problemas de contención de múltiples procesadores sobre un mismo bus, además de ofrecer un mejor rendimiento de manera más económica por el coste hardware, se desarrollaron los sistemas multicomputador, compuestos por múltiples unidades individuales con similar configuracion en capacidad de cómputo conectadas entre sí a través de una red.

La principal baza de estos sistemas es que la memoria en el conjunto es distribuida, contando cada unidad una porción de la memoria global a nivel privado para cada procesador, además de que para mantener la sincronización entre las diferentes unidades computacionales se emplea la tecnología basada en el paso de mensajes.

La tecnología de paso de mensajes se utiliza para sincronizar los distintos procesos entre las máquinas, tanto para compartir datos como para mantener correctamente la secuencia de la ejecución, esto sucede cuando los procesadores necesitan acceder a la información de otro procesador, o enviarse información entre ellos, ya que los datos no estan almacenados globalmente en el sistema y sí mas de un proceso necesita un dato, este debe duplicarse y ser enviado a todos los procesadores que lo necesiten.

El código del programa para cada procesador se carga en la memoria local al igual que cualquier dato que sea necesario, los programas todavía estan divididos en diferentes partes, como en un sistema de memoria compartida, y dichas partes se ejecutan concurrentemente entre los procesadores individuales.

La principal desventaja de este tipo de arquitectura, es que el límite se puede encontrar no en el ancho de banda de memoria local, sino más bien por la red de interconexión que une los diferentes computadores.

1.4.1 Arquitectura multicomputador cluster

La arquitectura basica de un sistema multicomputador cluster se basa en estar formado por un conjunto de ordenadores individuales (nodos) con un hardware similar llamadas interconectados entre sí a través de una red de comunicación más o menos rápida.

Cada nodo esta compuesto por una o varias unidades de procesamiento que direccionan con una memoria local en cada nodo, además de los canales de comunicacion de entrada/salida de los que pueda disponer para comunicarse con otras máquinas o el exterior.

La memoria local de cada nodo puede ser accedida por propio el procesador o procesadores del nodo, o por los procesadores externos a través de la red de interconexión. La memoria local en cada máquina puede usar el mismo direccionamiento de memoria dado que cada nodo

es un ordenador auto-contenido, donde el conjunto se podría considerar una arquitectura de mono/multiprocesadores interconectados con técnicas de paso de mensajes.

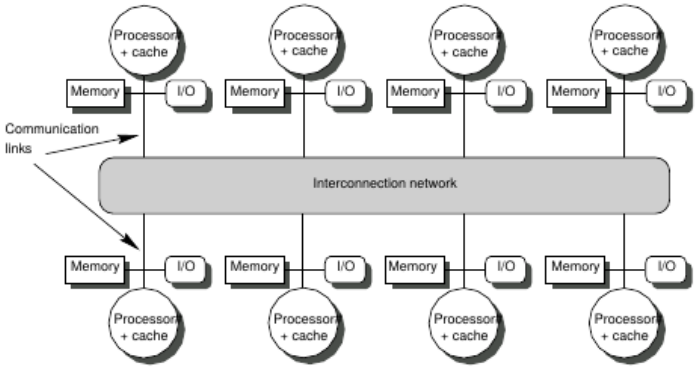


Figura 1.2: Diagrama de arquitectura multicomputador

El numero de nodos N dentro de este tipo de arquitectura puede ser tan pequeno como 16 (o menos), o tan grande como varios millares (o mas), sin embargo, la arquitectura de paso de mensajes (memoria distribuida) muestra sus ventajas sobre los sistemas de memoria compartida cuando el numero de procesadores es relativamente grande.

Para sistemas multicomputador con pocas unidades de cómputo, los sistemas de memoria compartida presentaran probablemente un mejor rendimiento y mayor flexibilidad debido a las velocidades más altas que puede permitir la interconexión. Además de que resultan más sencillos de programar.

Por contra, la ventaja de la arquitectura multicomputador es que es directamente escalable y presenta un bajo coste para integrar o ampliar sistemas grandes que buscan un incremento en alta potencia de cálculo.

En cada nodo del sistema, se ejecutan uno o más procesos de un programa, donde un proceso consiste a menudo en un código secuencial, como el que se encontraría en un ordenador Von Neumann. Sí existe más de un proceso en un procesador, este puede eliminarse del planificador cuando está a la espera de enviar o recibir un mensaje, permitiendo el inicio de otro proceso.

Se permite el paso de mensajes entre los distintos procesos de un procesador mediante el uso de canales internos o buses, en cambio, los mensajes entre procesos pertenecientes a diferentes procesadores se pasan a traves de canales externos usando la comunicacion existente entre los procesadores, a través de las redes de interconexión mediante mensajes.

Idealmente, los procesos y los procesadores que conforman el sistema pueden ser vistos como entidades completamente separadas.

Los problemas afrontados, usualmente se describen como un conjunto de procesos de cálculo que se comunican y sincronizan entre sí, encajandose sobre una estructura física de procesadores, por lo que el conocimiento de la estructura física y de la composición de los nodos es realmente recomendable y necesario a la hora de planificar una ejecución eficiente de una aplicación sobre este tipo de arquitecturas.

1.5 Limitaciones multicomputador cluster

Los arquitecturas multicomputador tienen mucho que ofrecer en relación a rendimiento, pero no son la solución infalible.

Existen limites en relación al número de ordenadores que se puede agregar e interconectar para mejorar el tiempo de ejecución de un problema, generalmente, se puede imaginar que la adición de más elementos de proceso tienen que reducir proporcionalmente el tiempo de ejecución de un programa, pero en la práctica no es así debido a algunos factores.

La combinación de mayor número de elementos de proceso produce un coste de rendimiento resultante de la sincronización entre tareas, comunicaciones y contención sobre los recursos compartidos, dicho coste tiende a crecer con el número de elementos de proceso y esto es un punto a tener en cuenta cuando se busca el rendimiento de la arquitectura.

Se suele suponer también, que un cálculo puede ser dividido en pequeños bloques de código capaces de ejecutarse completamente en forma paralela, pero mientras el código defina una parte que se ejecute de forma secuencial, esta parte limitará la paralelización de la ejecución en las distintas máquinas limitando también el rendimiento pico que se podría alcanzar en toda la arquitectura, es por ello que se deberían de mantener procesadores con suficiente potencia para resolver estas secciones de codigo existentes.

Además, existen algunas razones por las que algunos programas no pueden ser paralelizados completamente y deben ser ejecutados en un orden específico, el ejemplo más claro se basa en programas basados en entrada/salida, donde el orden de las operaciones está determinado por la disponibilidad, orden y formato de la entrada y el formato deseado de la salida.

Otra razón por la que la ejecución de programas pueden provocar limitaciones, se relaciona con las dependencias de datos, ya que sí el programa involucra variables que dependen de otras variables, el orden de las operaciones no se puede alterar sí no se disponen de los valores necesarios, por lo tanto se debe de asumir que se necesitan resolver estas dependencias durante la ejecución en un modelo de programacion paralelo y el coste temporal que puede generar en una arquitectura paralela.

Capítulo 2

Análisis de escenario

Los sistemas multicomputador cluster generalmente están restringidos a ordenadores individuales conectados en el mismo subrango de LAN trabajando de manera conjunta a disposición de la resolución de problemas complejos, siendo este uno de los principales usos a día de hoy en ámbitos de ciencia, investigación e ingeniería.

El siguiente proyecto está basado en la aproximación a una arquitectura multicomputador de tipo cluster orientada a alto rendimiento, evidentemente restringida en potencia por la limitación del hardware, donde el carácter del desarrollo de esta arquitectura es meramente orientado a ofrecer una plataforma apta para la investigación de algunos de sus principios e introducción a este tipo de sistemas.

En los siguientes apartados se tratará de aproximar un planteamiento inicial para el desarrollo de la arquitectura final.

2.1 Planificación

El desarrollo de un arquitectura multicomputador orientada a obtener el máximo rendimiento posible para que resulte una inversión eficiente, radica en gran parte a la fase de análisis de requisitos previos e investigación en las necesidades de la máquina a desarrollar posteriormente

Para aproximar el diseño del cluster, a continuación se destacan algunos puntos críticos para la planificacion del sistema que se vaya a necesitar, siendo muy importante determinar estos puntos antes de proseguir con la implementación de la infraestructura.

Algunos puntos que se deberían de considerar previamente serian:

- Determinar la misión del cluster
- Seleccionar una arquitectura para el cluster
- Seleccionar el sistema operativo
- Seleccionar el hardware para el cluster

El primero de los puntos es primordial, ya que determinará las principales necesidades que debería de resolver la infraestructura final. A partir de este punto se podría empezar a analizar

de que manera se podría afrontar un desarrollo que resolviera los requisitos decididos, por lo tanto será determinante sobre los puntos posteriores.

Es importante definir que se va a realizar con el sistema cluster, siendo este el primer paso del diseño, donde en un diseño en un entorno real se tendría que tener muy en cuenta el objetivo final para el que va destinado, si es un recurso abierto, para uno o distintos usos, cantidad de usuarios, etc, tratando de anticiparse siempre a cualquier posible conflicto que pueda surgir por necesidades, tanto hardware como software.

El segundo punto determinará la arquitectura óptima para que resulte eficiente aplicarla sobre los requisitos del primer punto. Esta fase del análisis incluiría el tipo de interconexión así como la distribución de la arquitectura óptima acorde las necesidades y requisitos establecidos.

El tercer punto implicará la selección del sistema operativo que intermediará para la utilización directa de la arquitectura. Existe una amplia gama de sistemas operativos utilizables, para este punto se deberá de elegir uno que aporte la funcionalidad y herramientas necesarias para cubrir las necesidades de la arquitectura.

Como último punto, la selección de hardware debería de ser el paso final del diseño, pero este punto puede reubicarse al principio si es necesario, sobretodo en caso de que solo se dispusiera de hardware antiguo o reutilizable o que se contara ya con material establecido previamente.

2.2 Misión del cluster

El primer punto del análisis se basa en la definición de los objetivos que se pretenden alcanzar con la arquitectura, definiendo así una misión principal para poder acotar algunos factores relacionados directamente con el sistema.

Tal como se describe en el apartado anterior, la parte hardware usualmente se define en última instancia si se trata de un proyecto que dependa de algún tipo de inversión o aplicacion determinada, o por el contrario que desde el principio el hardware este definido, de manera que la misión del cluster tenga que amoldarse a los recursos hardware de los que se disponga para realizar la configuración de todo el sistema.

En el caso expuesto en este proyecto, las limitaciones de la misión principal están acotadas inicialmente por el hardware utilizado, ya que los elementos físicos y hardware están establecidos desde el principio al basarse el desarrollo del sistema en material obtenido del reciclaje y de equipamiento en estado de desuso.

Por tanto, el objetivo de la implementación de este tipo de sistema en el ámbito del proyecto, está orientado principalmente a proporcionar un entorno didáctico que permita su utilización para la realización de pruebas de investigación a diferentes niveles, tanto en labores de administración como instalaciones o configuraciones, o en el testeo de herramientas y ejecución de programas. Además, este entorno proporciona una plataforma donde se podrán observar los principios de funcionamiento de los sistemas multicomputador sincronizados por paso de mensajes, permitiendo entrever algunos de los efectos derivados de la ampliabilidad, rendimiento, limitaciones, análisis, etc.

2.3 Hardware

El punto de la definición del hardware que se utilice para el sistema puede abordarse desde dos puntos iniciales:

- Proyecto de diseño completo
- Hardware predefinido: reutilización de material, reciclaje, desuso

En el primer punto, la selección de hardware se aproximaría a los últimos puntos de diseño, necesitándose previamente un proceso de análisis de requisitos y diseño complejo para obtener el mayor rendimiento y eficiencia posible para aprovechar una inversión económica.

En el segundo punto, el hardware ya está definido desde el primer momento, por lo que el disponer de el desde el principio ya acotara su utilizacion si existen limitaciónes dependiendo del tipo de equipamiento del que se disponga.

En el desarrollo de este proyecto, el hardware está definido desde el principio, por lo que la adecuación del entorno y la implementación deberán de adaptarse al tipo de equipamiento disponible. El hardware ha sido obtenido del reciclaje de material que estaba ya en desuso, disponiendo de diferente equipamiento no homogéneo, destacando el hecho de contar con 7 ordenadores en formato desktop antiguos, por lo que las características hardware no contarán con los últimos desarrollos en tecnología de vanguardia, definiendo ya algunas limitaciones en términos de rendimiento/eficiencia desde el principio.

El hardware final seleccionado entre el equipamiento para la implementación del sistema será el siguiente:

- 1 equipo con modelo de procesador Intel Dual Core, con compatibilidad para instrucciones 32 y 64 bits
- 4 equipos con dos modelos de procesadores Intel Pentium 4 donde el factor común es que operan comúnmente en instrucciones de 32 bits
- Tarjetas Ethernet con conexión PCI/Express, operativas todas a velocidades de 10/100 MBPs
- Reposiciones de discos duros
- Memoria RAM DDR y DDR2, en diferentes gamas y velocidades
- Switch de interconexión Gigaethernet domestico
- Periféricos de entrada salida (ratones, teclados, pantallas)
- Diferentes tipos de cableado
- Artículos para adecuación de cables/entorno

2.4 Arquitectura

El punto relacionado con la arquitectura, basándose en la arquitectura multicomputador deseada, depende directamente en el caso aquí expuesto del hardware disponible inicialmente. En este caso, se dispone desde el primer momento de ordenadores desktop, PCS comunes de sobremesa antiguos obtenidos de contenedores de reciclaje, además de hardware de red y otras piezas hardware y consumibles.

Así pues, la arquitectura sobre la que se basará la implementación del sistema cluster se conoce como **Beowulf**, donde la característica principal en este tipo de arquitectura es que está basada en la arquitectura convencional multicomputador pero implementada con hardware estándar no especifico a través de una interconexión LAN privada para los distintos equipos.

Este tipo de arquitecturas usualmente ejecutan un sistema operativo de libre distribución, principalmente basado en Linux, BSD o Solaris, donde opera sobre los diferentes componentes del sistema.

Debido a que los componentes principalmente son grupos de equipamiento estándar o estaciones de trabajo **Workstations**, sumando la utilización de un software de libre distribución y usualmente gratuito, este tipo de configuración ofrece una solución para obtener mayor capacidad de cálculo a un bajo coste y ofreciendo flexibilidad en caso de necesitar ampliaciones hardware o mantenimiento.

El nombre de este tipo de arquitectura cluster hace referencia a un computador construido en 1994 por Tomhas Sterling y Donal Becker en la NASA, donde el nombre proviene de un antiguo poema ingles.

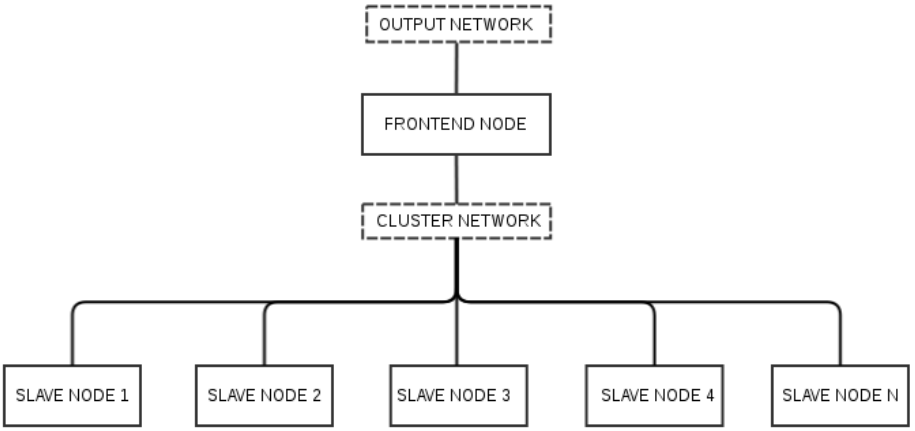


Figura 2.1: Arquitectura de un Beowulf cluster

2.5 Sistema operativo

Como fase final en el desarrollo del cluster, se deberá de seleccionar el sistema operativo que operará sobre la infraestructura hardware. Esta selección está basada principalmente en la compatibilidad con el hardware disponible, aunque también influyen factores como la preferencia personal por algún tipo de sistema operativo en concreto, coste, etc.

En el ámbito de este proyecto, uno de los factores principales es la utilización de un sistema operativo basado en **GNU/Linux**, básicamente por algunas de las ventajas que pueden ofrecer algunas distribuciones basadas este sistema en el desarrollo del entorno multicomputador, ademas, supone un coste nulo y se cuenta con mucha documentación al respecto.

En este punto, partiendo de la utilización de un sistema GNU/Linux, es interesante conocer la existencia de cluster kits, que son distribuciones compuestas por conjuntos de paquetes software que facilitan el proceso de instalación de un sistema cluster, ofreciendo utilidades y herramientas para su utilización y administración. Normalmente estos kits suelen ser muy completos a nivel de software, simplificando además su instalación para obtener la implementación de un cluster, permitiendo ponerlo en funcionamiento de manera simple, enfocando principalmente al usuario y a la utilización del software aplicativo. Como problemática, la utilización de estos kits implica que parte de la configuración viene simplificada, por lo que un usuario poco familiarizado con sistemas basados en GNU/Linux puede tener problemas si modifica alguna configuración sin saber exactamente que se está haciendo.

Exponer, que aunque usualmente es tendencia natural el utilizar las últimas versiones disponibles de un sistema donde normalmente existen mejoras y correcciones, en algunas ocasiones se prima por la compatibilidad, por lo que la decisión de elegir una versión puede llevar a seleccionar una más antigua para asegurarse de que es totalmente compatible con el hardware disponible.

También, en versiones antiguas de releases, la mayoría de los problemas son conocidos y están documentados, además de ofrecer más compatibilidad de drivers cuando se utiliza hardware obsoleto o discontinuado. Por su contra, las versiones antiguas pueden contener posibles fallos de seguridad conocidos, por lo que es posible que sean más expuestos a sufrir afecciones por malware o intrusiones si no están correctamente corregidos.

2.5.1 Selección

Basando la selección del sistema operativo en el hardware disponible, donde los factores comunes que comparten son la antigüedad y la arquitectura de 32 bits, y teniendo en cuenta la utilización de un kit que facilite la configuración básica inicial, se decanta la selección del sistema operativo a la utilización de la distribución Rocks Cluster basada actualmente en CentOS, después de selección final de entre otras distribuciones disponibles.

A continuación, se facilitan algunos enlaces relacionados con este sistema operativo:

- **Pagina Oficial** `http://www.rocksclusters.org/wordpress/`
- **Descarga** `http://www.rocksclusters.org/wordpress/?page_id=80`
- **Distrowatch** `https://distrowatch.com/table.php?distribution=rockscluster`

2.5.2 Rocks Cluster

Rocks Cluster es una distribución orientada a High Performance Computing desarrollada por el National Partnership for Advanced Computational Infrastructure y por el Sandiego Supercomputer Center (SDSC). Inicialmente estaba basada en Red Hat Linux, pero actualmente su núcleo está basado en CentOS, con un instalador modificado (Anaconda) que simplifica el despliegue inicial en múltiples máquinas.

La principal ventaja que ofrece es que desplegar un sistema cluster con esta distribución no requiere de una alta experiencia en este tipo de sistemas, ya que se ofrece una solución flexible, sencilla y cómoda para implementar la configuración del software base. Además, la distro cuenta con una amplia variedad de soporte y documentación a través de la red de Internet, tanto en foros como en canal activo de correo.

La instalación de este sistema se basa en un núcleo principal y paquetes de software modulares instalables llamados Rolls, extendiendo la funcionalidad dependiendo de las necesidades del escenario, ya que se integran de manera automática en los mecanismos de control usado por el software base, simplificando en gran medida el proceso de instalación.

El sistema Rocks genera una base de datos con la configuración existente en el cluster, donde a partir de esta se generan posteriormente los diferentes ficheros de configuración para su funcionamiento.

Además, la gestión de configuraciones internas del sistema operativo así como la instalación de software nuevo y sincronización entre los diferentes computadores, está ampliamente simplificada, facilitando la administración por parte del usuario o por el administrador del sistema.

Algunas de sus características principales son:

- **OS Type:** Linux
- **Basado en:** CentOS
- **Origen:** USA
- **Arquitectura:** x86_64, heterogéneo
- **Escritorio:** Gnome - KDE
- **Categoría:** Clusters, HPC
- **Estado:** Activo
- **Modelo:** Open Source
- **Último release:** 6.2 (Sidewinder) / May 12, 2015
- **Tipo Kernel:** Monolítico

El abanico de hardware soportado por Rocks Cluster es verdaderamente amplio desde los primeros releases bajo la distribución Red Hat, pero solo es capaz de soportar arquitecturas x86, x86_64 y arquitecturas IA-64. Entre el hardware principal soportado por la distribución se puede encontrar:

Procesadores:

- x86 (ia32, AMD Athlon, etc.)
- x86_64 (AMD Opteron and EM64T)
- IA-64 (Itanium)

Tecnologías de redes e interconexión:

- Ethernet(incluido Intel Gigabit Ethernet)
- Myrinet (Myricom)
- Infiniband (Voltaire)

2.5.3 Requisitos mínimos instalación

Como cualquier sistema, se necesitan unos requisitos mínimos de hardware necesarios para realizar la instalación de la distribución Rocks, se listan a continuación:

	Disco	Memoria	Interfaces red
Frontend Node	20GB	512MB	2 puertos físicos

Tabla 2.1: Requisitos hardware mínimos para instalación del nodo central

	Disco	Memoria	Interfaces red
Compute Node	20GB	512MB	1 puertos físicos

Tabla 2.2: Requisitos hardware mínimos para instalación de un nodo cálculo

2.6 Entorno

El entorno es una parte importante en la implementación de un sistema cluster, ya que si se tratara de una instalación donde en el futuro se proyectara algún tipo de ampliación se debería de contar con espacio y algunas condiciones determinadas.

Algunas de las recomendaciones a nivel de entorno para el despliegue podrían ser:

- Ubicación de seguridad a nivel físico del hardware
- Suministro de electricidad adecuado
- Ventilación correcta para mantener una temperatura de funcionamiento adecuada
- Espacio para posibles ampliaciones

Debido a que el cluster desplegado en este proyecto cuenta con pocas máquinas en formato estándar, estas variables no deberían de afectar en gran medida, aun así, algunos detalles como la disposición de los distintos ordenadores móvil para facilitar posteriormente el acceso para mantenimiento o la refrigeración por circulación de aire entre las máquinas deberían ser puntos a tener en cuenta.

Para prevenir sobrecalentamiento, se retirarán todas las tapas que se puedan en nuestro caso para facilitar la refrigeración de los ordenadores y así permitir una circulación de aire correcta, la disposición se realizara de manera contigua entre las carcasas de los equipos.

El cableado desplegado se tratará de acomodar unido dependiendo de su función (eléctrico o interconexión) para facilitar en un futuro mantener un orden en caso de mantenimiento o ampliación.

La ubicación del equipamiento en este caso se realizará en una habitación pequeña, con espacio reducido, por lo que el aprovechamiento de una zona se complementará con una plataforma móvil para poder desplazar el sistema cuando no se utilice.

Como medida preventiva para posibles sobrecalentamientos por uso continuado se cuenta con un ventilador pequeño de categoría industrial.

Capítulo 3

Implementación Cluster Rocks

Tal como se ha comentado brevemente en el capítulo anterior, la ventaja del uso de la distribución Rocks para crear y mantener un cluster es simple: la configuración inicial de un cluster puede resultar sencilla, pero la gestión del software necesario puede ser más compleja.

Esta complejidad puede llegar a resultar ingestionable cuando se habla de la instalación o la expansión de un cluster o sistema multicomputador, para ello el sistema operativo Rocks proporciona mecanismos de control más amigables con el proceso de instalación y expansión de la arquitectura.

A continuación se detalla el proceso de conformado del sistema cluster, pasando tanto por su fase de ensamblado físico como por la instalación del sistema operativo en los diferentes equipos.

3.1 Hardware principal

Uno de los puntos más importantes en que se basará la implementación del cluster es la parte física, el hardware. En el caso de este proyecto es un punto determinante ya que el hardware está definido desde el principio contando con unas máquinas concretas disponibles y la compatibilidad que ofrecen.

A continuación, se detallarán las partes hardware más importantes en la implementación de la arquitectura cluster: procesadores, memoria e interconexión.

3.1.1 Procesadores: Intel Pentium 4

El procesador es una de las partes más importantes dentro del sistema cluster, gran parte del rendimiento de la máquina en conjunto recaerá sobre el tipo de procesador que se haya instalado en las diferentes máquinas que conformen la arquitectura, ya que la función principal es ejecutar los diferentes programas que se le asignen aportando la capacidad de cálculo que disponga individualmente al sistema en su conjunto.

En el caso expuesto, al tratarse de máquinas recicladas antiguas aprovechadas para la realización de este proyecto, la selección de este hardware se limita a los diferentes procesadores disponibles montados con anterioridad en los equipos. En un proyecto real basado en la implementación de un sistema multicomputador, se necesita realizar un estudio previo, pues la elección de este componente es clave en función de la necesidad y los objetivos que se pretendan alcanzar con la

arquitectura. Advertir de que una mala elección puede perjudicar en gran medida el rendimiento final del sistema, así que es un punto muy a tener en cuenta que tiene que someterse a estudio y valoración.

En el sistema implementado, los diferentes ordenadores disponibles montaban varios modelos de procesadores, todos de la marca Intel, donde para el proceso de selección de equipos se ha tratado de homogeneizar las características de los elegidos, seleccionando los más similares entre sí para evitar posibles descompensaciones en el rendimiento.

Node	Modelo
Frontend	Intel Core 2 Duo E4300
Compute-0-0	Intel Pentium 4 (Northwood)
Compute-0-1	Intel Pentium 4 (Northwood)
Compute-0-2	Intel Pentium 4 (Prescott 2M)
Compute-0-3	Intel Pentium 4 (Prescott 2M)

Tabla 3.1: Procesadores utilizados en los distintos nodos del cluster

- Para el rol de los 4 nodos computacionales, se dispondrán procesadores Intel Pentium 4 en dos familias distintas Northwood y Prescott 2M. Estos procesadores serán los encargados de realizar el procesamiento bruto de la computación.
- Para el rol de nodo de gestión o frontend, se contará con un procesador Intel Core 2 Duo, que no entrará a formar parte en ningún momento del desempeño computacional del cluster pero al contar con más potencia facilitara la gestión y la utilización de algunas utilidades posteriores o sesiones multiusuario.

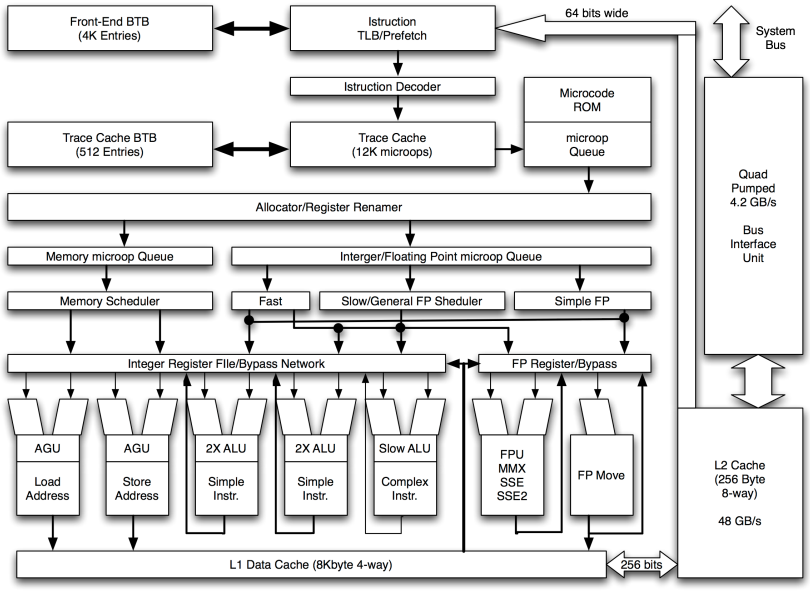


Figura 3.1: Arquitectura procesador Pentium 4

Como detalle de la arquitectura de los procesadores que se montarán en los nodos computacionales, el Pentium 4 fue la séptima generación de Intel basada en arquitectura i686, llamada NetBurst (P68). Este procesador es el último IA-32 preparado con el set completo de instrucciones IA-32 SIMD, de carácter CISC.

Su conjunto de instrucciones es una extensión de la ISA utilizada en la versión anterior Pentium 3, donde incluye SSE2 (Streaming SIMD Extensions 2) para extender la capacidad de la tecnología Intel MMX. Este nuevo set de instrucciones incluye:

- 144 nuevas instrucciones para trabajar con operaciones aritméticas de enteros en 128-bit SIMD.
- Doble precisión en 128-bit SIMD para operaciones de coma flotante FP.

La microarquitectura NetBurst perseguía los siguientes objetivos:

- Ejecutar IA-32 y aplicaciones SIMD.
- Operar con altos valores de frecuencia con posibilidad de escalabilidad en el futuro.

Los procesadores utilizados en el sistema cluster están englobados dentro de la familia número 15 del productor Intel, caracterizados por la arquitectura descrita anteriormente con byte order Little Endian. Las características principales de los procesadores utilizados en el sistema son las siguientes:

	Intel Pentium 4 2.40 GHz	Intel Pentium 4 620	Intel Core2Duo E4300
Code Name	Northwood	Prescott 2M	Conroe
Lithography	130 nm	90 nm	65 nm
Cores	1	1	2
Base Frequency	2.40 GHz	2.80 GHz	1.80 GHz
Cache	512 KB L2	2 MB L2	2 MB L2
Bus Speed	533 MHz FSB	800 MHz FSB	800 MHz FSB
TDP	59.8 W	84 W	65 W
Instruction Set	32-bit	32_64-bit	32_64-bit

Tabla 3.2: Características de los modelos de CPU utilizados en el cluster

Además, los procesadores englobados dentro de los modelos seleccionados de Pentium 4 cuentan con las siguientes tecnologías:

- Hiper-Pipelined Technology
- Advanced Dynamic Execution
- Rapid Execution Engine
- Execution Trace Cache
- 533 MHz Front Side Bus
- Advanced Transfer Cache
- Streaming SIMD Extensions 2 (SSE2) instructions
- Hyper-Threading Technology

3.1.2 Memoria RAM

La memoria RAM como componente define el espacio de almacenamiento temporal donde se alojará el código de instrucciones de los programas ejecutados en el sistema cluster.

Tal como se ha descrito anteriormente, en un sistema multicomputador por paso de mensajes, cada máquina a nivel individual dispone de un espacio de memoria privado direccionado de manera local por su propio procesador/es y distribuido a nivel global de sistema.

Debido a la antigüedad del hardware con el que se cuenta para el desarrollo del proyecto, la tecnología de memoria RAM cuenta con tecnologías de hace algunos años, limitando algunas de las ventajas que se disponen actualmente como mayor integracion de capacidad, velocidades o latencias.

A continuación se resumirán algunas de las características principales con las que cuenta la memoria utilizada en el sistema:

Node	Capacidad	Modelo	Tipo	Frecuencia	Latencia
Frontend	2GB	KVR800D2N6/2G	DDR2 800	800 MHz	CL6
Compute-0-0	1GB	KVR400X64C3EA	DDR400 (PC3200)	400MHz	CL3
Compute-0-1	1GB	KVR400X64C3EA	DDR400 (PC3200)	400MHz	CL3
Compute-0-2	2GB	KVR667D2N5	DDR2 (PC25300)	667 MHz	CL5
Compute-0-3	1GB	KVR400X64C3EA	DDR400 (PC3200)	400MHz	CL3

Tabla 3.3: Memoria utilizada en los distintos nodos del cluster

3.1.3 Hardware de interconexión

Uno de los puntos más importantes dentro del diseño e implementación de una arquitectura multicomputador se basa en la tecnología empleada en la interconexión de las distintas máquinas que conforman la arquitectura. Al tratarse de una arquitectura que basa su funcionamiento en la sincronización por paso de mensajes, disponer de un canal suficientemente rápido para permitir la agilidad de comunicación entre los procesadores es de vital importancia para garantizar el rendimiento final del conjunto.

En el caso expuesto, para la interconexión de red de la implementación, se dispone un switch doméstico auto-gestionable con tecnología GigabitEthernet. Es evidente que este tipo de dispositivo no contaría con un grado de idoneidad frente a un proyecto de mayores requisitos, donde se necesitaría hardware más específico y tecnologías de interconexión más rápidas, pero será funcional para el proyecto desarrollado.

El modelo del switch utilizado se trata del modelo: **D-Link DGS-1008D**

Las principales características de este switch son las siguientes:

- Standars IEEE 802.3, 802.3u
- 8 puertos 10/100/1000 Mbps
- Auto-uplink MDIII/ MDI-X
- Auto-negociación full / half duplex
- Control de flujo

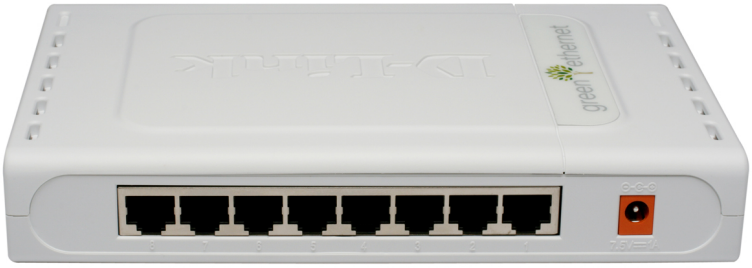


Figura 3.2: D-Link DGS-1008D

A continuación se detallan los estándares con los que es compatible:

- IEEE 802.3 10BASE-T
- 802.3u 100BASE-TX
- 802.3ab 1000BASE-T
- 803.3x control de flujo.

Algunas de las características extra de las que dispone el dispositivo de red son las siguientes:

- **Control de flujo:** El Switch DGS-1008D, en modo Full-Dúplex, permite proteger a los usuarios frente a posibles pérdidas de datos durante la transmisión en la red. Cuando están conectados a una tarjeta (en un servidor o PC) que soporte control de flujo y cuando el buffer de datos está lleno, el switch envía una señal al PC indicando la situación para que el equipo demore la transmisión hasta que el buffer se haya liberado y sea posible el envío de más información.
- **Conectividad Gigabit sin fibra:** El switch dispone de 8 puertos GigabitEthernet, por lo que funciona sin problemas con los cables de red existentes (Cat5 o superior). En modo full/duplex pueden alcanzarse hasta 2000 Mbps, con lo que las estaciones de trabajo podrán trabajar sin cuellos de botella.
- **Instalación plug and play:** Todos los puertos admiten tanto cables straight-through como crossover, lo que elimina la necesidad de disponer de un puerto uplink. Dados los 10/100/1000 Mbps auto-gestionables, no se requiere configuración alguna.
- **Tecnologías de ahorro energético D-Link Green:** Incorpora tecnologías de ahorro de energía. Al detectar sí los dispositivos conectados están encendidos o apagados y al estimar la longitud del cable, el switch es capaz de ajustar su consumo energético, lo que supone un ahorro de hasta el 73 %.

Comentar que la velocidad ofrecida por el switch se vera reducida, debido a que las tarjetas de interconexión que se cuenta en los equipos utilizan tecnología antigua, trabajando a velocidades de 10/100 MBps. Esta situación debería de corregirse para garantizar la máxima velocidad de interconexión, pero no se dispone de otro hardware, por lo que es posible que esto afecte negativamente al rendimiento del sistema.

3.1.4 Sumario Ganglia

A continuación se detallan los sumarios de cada computador implicado en el sistema, obtenidos con el software de monitorización Ganglia que se detallará en los siguientes capítulos.

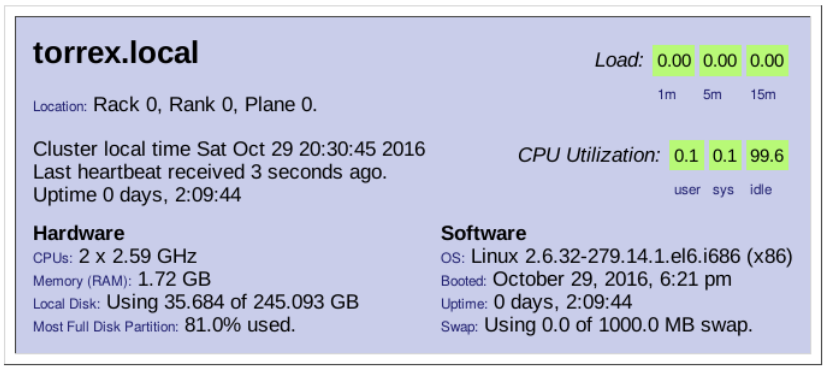


Figura 3.3: Sumario de recursos hardware/software en nodo control frontend

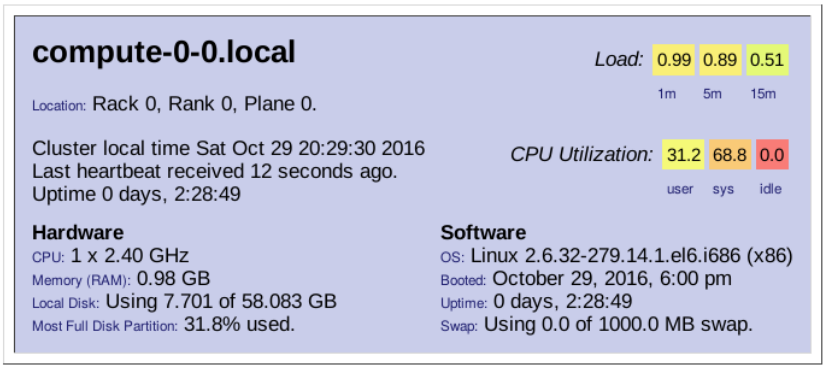


Figura 3.4: Sumario de recursos hardware/software en nodo cálculo 0

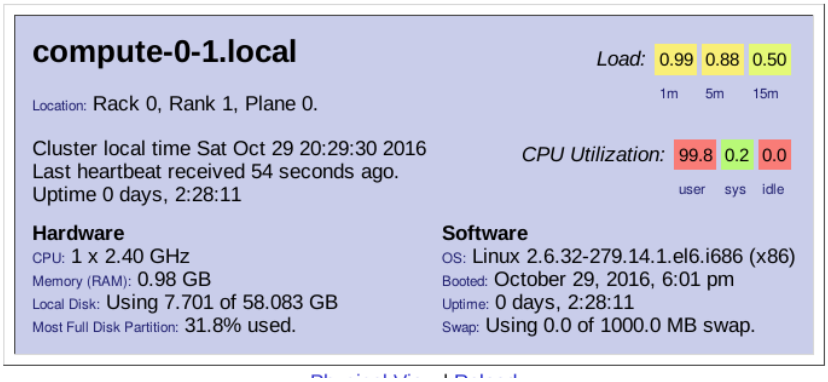


Figura 3.5: Sumario de recursos hardware/software en nodo cálculo 1

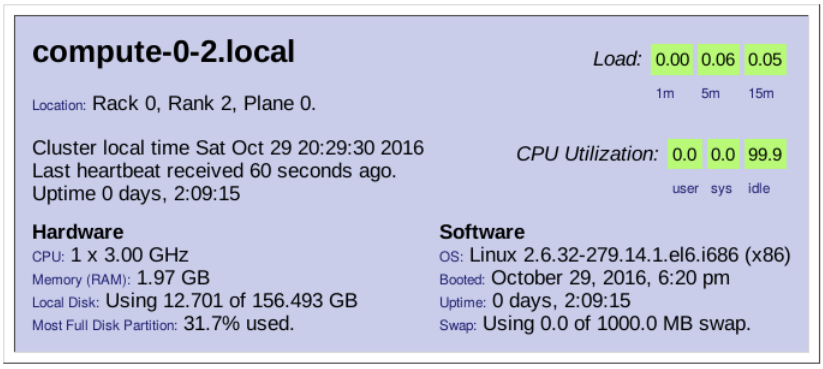


Figura 3.6: Sumario de recursos hardware/software en nodo cálculo 2

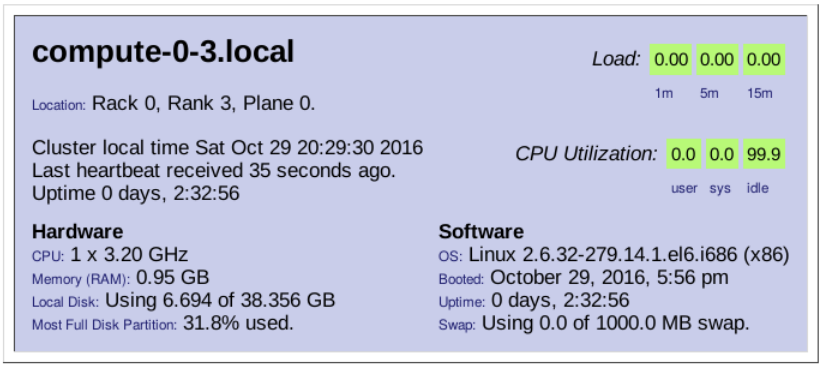


Figura 3.7: Sumario de recursos hardware/software en nodo cálculo 3

3.2 Interconexión: red de comunicación

La red de interconexión, como se ha comentado anteriormente, es uno de los elementos mas importantes en una arquitectura multicomputador puesto que puede influir directamente en el rendimiento global del sistema.

La red de interconexión será el medio por el que se van a comunicar los diferentes elementos que conformen el sistema, tanto para transferencia de datos como para paso de mensajes de sincronización entre los diferentes nodos. Es por ello que la velocidad y la disponibilidad de este medio tiene que ser un punto a tomar en cuenta sí se desarrolla un sistema multicomputador de altas prestaciones.

Existen diferentes tipos de tecnologías de interconexión, así como diferente hardware para unir los elementos del cluster, la selección de estos elementos tiene que someterse a un análisis previo para finalmente adecuar una solución que solvente las problemáticas y los objetivos que tenga que afrontar el sistema.

En el caso expuesto, el condicionamiento de disponer de material reutilizado para la implementación del cluster, limita directamente la garantía de trabajar con la máxima velocidad ofrecida por el hardware, ya que coexisten diferentes velocidades de interconexión entre el switch y las tarjetas de red del equipamiento. Tal como se ha descrito, esto puede suponer una merma de rendimiento en sistema por lo que en el caso de adquirir este hardware partiendo de cero, se debería de prestar especial atención a este punto.

La distribución de los elementos así como la interconexión que los vincula entre ellos, generará una figura que determinará algunas características como la redundancia, la alta disponibilidad o el direccionamiento. En la siguiente sección se describe este punto.

3.2.1 Topología

Dependiendo de la interconexión que se realice entre los diferentes elementos se pueden obtener diversas características.

El componente comun dentro de la interconexión de un multicomputador se basa en una serie de elementos que distribuyan la comunicación entre los diferentes nodos del cluster, permitiendo la visibilidad deseada entre ellos.

En el caso implementado en este proyecto, la conexión entre nodos se realiza mediante el hardware de distribución de red (switch), donde la conexión se realiza de forma directa entre nodo y switch con una única línea de comunicación mediante cable, intentando garantizar la máxima velocidad. Esta línea de comunicación no incluye líneas de comunicaciones directas con otros nodos ni redundancia implícita, por tanto, en este tipo de conexión se podría destacar que tanto la comunicación necesaria en la computación como la comunicación de gestión estarán solapadas dentro del mismo medio, donde para evitar esto se podría incluir una red alternativa para segmentar la interconexión, en caso de disponer de más hardware.

El diagrama de interconexión de red y la topología generada se muestra en la siguiente figura:

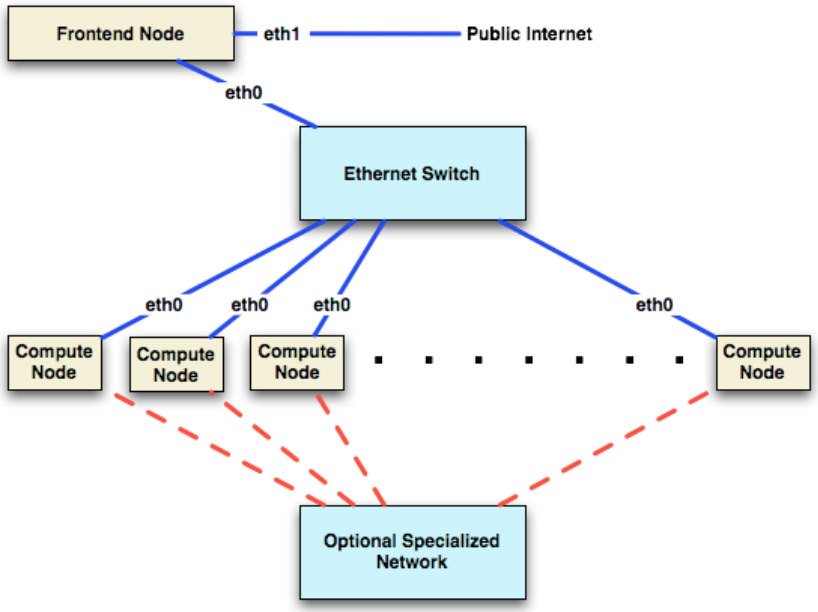


Figura 3.8: Diagrama de interconexión de los elementos del cluster

Sí se tiene en cuenta la red establecida en el montaje, la topología generada sería una estrella, donde todos los nodos o elementos de computación cuelgan directamente del nodo de gestión o frontend, utilizando como elemento de interconexión un solo switch en este caso.

Hay que tener en cuenta que la ampliabilidad es una característica importante en el diseno de sistemas multicomputador, donde se debería de facilitar la adición de más elementos al sistema además de mantener un rendimiento óptimo del conjunto. Para cumplir este objetivo, la arquitectura facilita este punto, donde en el proyecto expuesto en caso de agotar las conexiones

del switch se podría utilizar más hardware de interconexión interconectado entre sí, permitiendo la adición de más elementos de cómputo de manera flexible y cómoda.

Es importante destacar, que en caso de ampliar el numero de nodos en el sistema, el ancho de banda total de comunicaciones, memoria y capacidad de procesamiento del sistema tambien aumenta, pero en el caso de contar con una tecnología no apropiada de interconexión esta saturación puede degradar el rendimiento final de todo el conjunto.

3.2.2 Direccionamiento

El direccionamiento IP de los elementos del cluster es un punto importante, ya que el funcionamiento conjunto del sistema se basará en que todos los nodos sean visibles entre ellos para poder compartir información mediante paso de mensajes. Para permitir el acceso al exterior de la red del cluster, el nodo maestro dispondrá de dos interfaces de red, donde se definirá una dirección interna privada para los diferentes nodos de cálculo y una dirección externa "pública"para la conexión directa con la red de Internet o Intranet.

En los nodos de cálculo, el direccionamiento asignado se basará en la dirección privada definida en el nodo maestro, donde a través de DHCP se desplegarán posteriormente direcciones a cada solicitud de inserción de un nuevo nodo.

Las diferentes IP's asignadas a los diferentes nodos están definidas en la siguiente tabla. Se han obtenido tras la instalación completa del sistema.

Nodo	Descripción	Dirección IP Interna	Dirección IP Externa
Frontend	Equipo central maestro	10.10.1.1	192.168.1.69
Compute-0-0	Equipo de cálculo 1	10.10.1.254	———
Compute-0-1	Equipo de cálculo 2	10.10.1.253	———
Compute-0-2	Equipo de cálculo 3	10.10.1.252	———
Compute-0-3	Equipo de cálculo 4	10.10.1.251	———

Tabla 3.4: Direccionamiento IP de los elementos del sistema una vez instalados

3.3 Instalación Cluster

En la siguiente sección se detallará el procedimiento para ensamblado, puesta en marcha, instalación y configuración básica del cluster, donde dentro de este proceso se tratarán de englobar los siguientes hitos en el proceso de implementación.

1. Ensamblado y conexión de máquinas.
2. Instalación de nodo principal Frontend.
3. Instalación de nodos de cómputo.
4. Configuración post-install, entorno y utilidades.
5. Test de funcionamiento.

3.3.1 Ensamblado físico

Para poder empezar a desarrollar el cluster es necesario desplegar los diferentes equipos ensamblados correctamente en el espacio que se haya definido y realizar todas las conexiones que se necesiten, tanto a nivel de alimentación como de interconexión de red.

La interconexión entre ellos seguirá la distribución establecida en la topología, donde los nodos de computación están directamente conectados a través de un switch al nodo maestro o frontend. En caso de instalaciones de gran cantidad de nodos es altamente recomendable identificar los cables utilizados para la interconexión.

En la situación del proyecto, se dispone de equipos en formato desktop, por lo que una posible distribución es alinearlos entre las diferentes torres o apilarlos, para procurar que posteriormente el cableado esté lo más ordenado posible. Por la ubicación de la instalación en el caso expuesto, los diferentes equipos se han distribuido de forma contigua sobre una plataforma de madera con ruedas para facilitar su desplazamiento y acceso para mantenimiento.

En el siguiente diagrama se muestra el esquema del sistema final, diferenciando las dos redes implicadas así como la interconexión de red que se debería de realizar para interconectar los diferentes nodos de cálculo dentro del cluster.

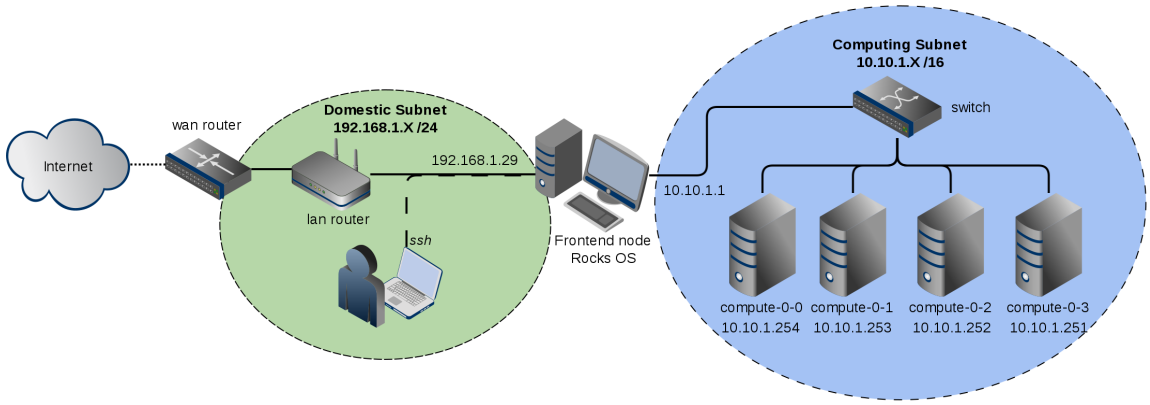


Figura 3.9: Diagrama de la distribución de conexiones de los distintos elementos diferenciado a nivel de subred

Al realizar la instalación del frontend, se debería de tener en cuenta que por defecto el sistema operativo Linux mapea las unidades con nombre eth*, en los nodos de cálculo al disponer de solo una interface de red está será mapeada como eth0, por ello, en el caso del nodo maestro, se debería de distinguir que el interface mapeado como eth0 debe de coincidir con la conexión al switch central de distribución, ya que será en esta interface donde se desplegará la IP privada de la LAN de cómputo, y en el caso de la interface eth1, donde se conectará la red externa de la instalación para garantizar su acceso remoto, conexión a Internet, etc.

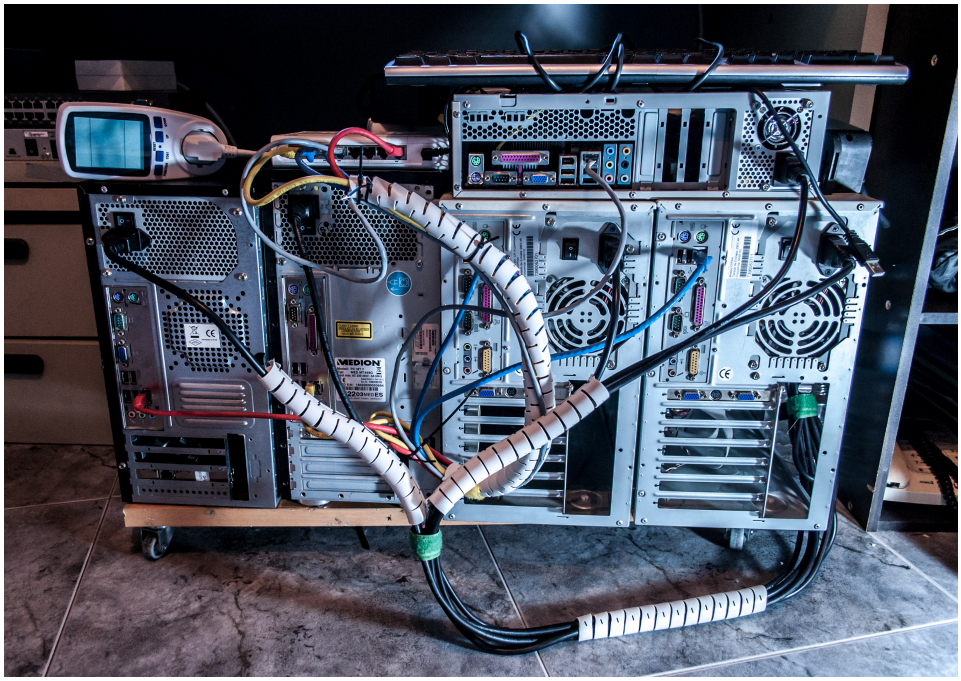


Figura 3.10: Visión trasera de las unidades de ordenadores que conformarán el sistema cluster completamente conectadas y ubicadas sobre plataforma móvil.

3.3.2 Instalación del Frontend

La instalación del nodo de gobierno o Frontend, es el primer paso una vez estén ensambladas las distintas máquinas e interconectadas correctamente entre sí. Para proceder al proceso de instalación se necesitarán los soportes de la distribución descargados de la página oficial.

Los Rolls mínimos requeridos para la instalación de la distribución Rocks Cluster en el frontend son los siguientes:

- **Kernel Roll** - distribución de kernels booteables
- **Base Roll** - roll base del sistema
- **HPC Roll** - herramientas software orientadas a programación paralela
- **OS Roll** - sistema operativo CentOS

Como Rolls complementarios se podrían destacar:

- **Area 51 Roll** - servicios y herramientas de seguridad
- **Ganglia Roll** - sistema de monitorización del cluster
- **SGE Roll** - Sun Grid Engine, sistema de encolado de trabajos

Estos medios se pueden descargar en formato DVD, siendo este el indicado porque contiene todos los rolls disponibles para la distribución. Este formato por tanto, facilita la instalación del software necesario ofreciendo un punto importante en la simplificación del despliegue.

Para realizar la instalación, se debería de configurar en la BIOS del equipo Frontend el orden del arranque (Boot Priority) en el medio donde ubiquemos la fuente descargada (DVD/USB).

Una vez iniciado el medio del instalador, la pantalla mostrará diferentes opciones, la opción **build** inicia el asistente instalador en modo Frontend, tal como se visualiza en la siguiente captura.

Figura 3.11: Instalación Rocks: Pantalla de inicio

El botón CD/DVD-based rolls, mostrará la pantalla con los diferentes rolls que se permiten instalar desde el medio extraíble. Para instalarlos en el sistema basta con seleccionar los que se consideren necesarios para proseguir con la instalación del nodo Frontend.

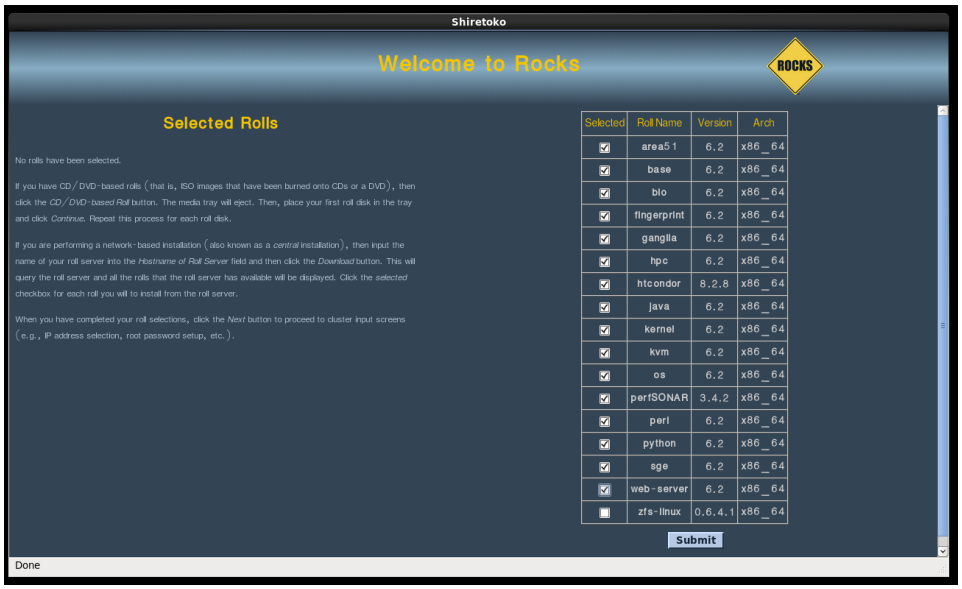


Figura 3.12: Instalación Rocks: Selección de rolls a instalar

A continuación, se podrán comprobar la lista de los Rolls seleccionados, contando que generalmente cada Roll aporta un software específico para un tipo de tareas, una vez comprobados y seleccionados se puede proceder a su instalación mediante el botón Next.

Figura 3.13: Instalación Rocks: Rolls seleccionados

En la siguiente pantalla, se deberá de introducir la información identificativa referente al cluster que se está instalando. Los campos que se deberían de completar son los siguientes:

- **Fully-Qualified Host Name** - Nombre de dominio
- **Cluster Name** - Nombre del cluster general
- **Certificate Organization** - Organización a la que pertenece el sistema
- **Certificate Locality** - Localidad
- **Certificate State** - Provincia
- **Certificate Country** - País
- **Contact - Email** de contacto
- **URL** - Dirección web relacionada al cluster
- **Latitude/Longitude** - Ubicación

Esta información se utiliza sí se desea registrar el cluster en la clasificación de Rocks, donde se pueden introducir datos relacionados a rendimiento para comparaciones posteriores.

En la página oficial de Rocks Cluster existe una clasificación basada en la potencia de cálculo de los sistemas registrados, mostrando la información referente a cada uno basada en el tipo de procesador utilizado, número de procesadores, frecuencia, FLOPS o ubicación.

La clasificación se puede consultar en el siguiente enlace:

`http://www.rocksclusters.org/rocks-register/`

Figura 3.14: Instalación Rocks: Información identificativa del cluster

En las siguientes pantallas se configuran los adaptadores de red, el **externo (eth1)** para la conexión con red corporativa - Internet, y el **interno (eth0)** utilizado para la red privada donde se ubican los nodos de cálculo.

El direccionamiento se completará basandose en en el apartado donde se ha descrito con anterioridad.

Nodo	Descripción	Dirección IP Interna	Dirección IP Externa
Frontend	Equipo central maestro	10.10.1.1	192.168.1.69

Tabla 3.5: Direccionamiento IP del nodo central Frontend

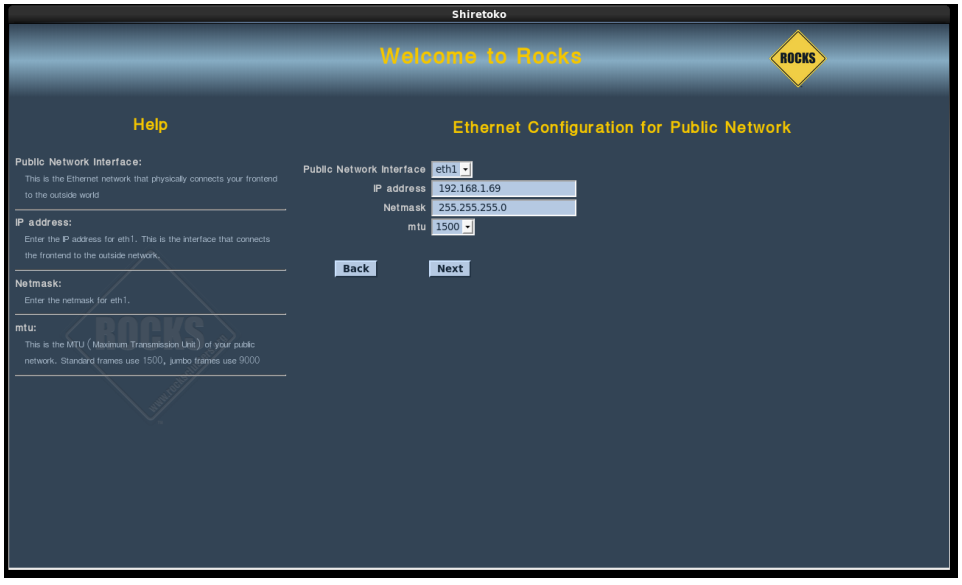


Figura 3.15: Instalación Rocks: Configuración interfaz de red eth1

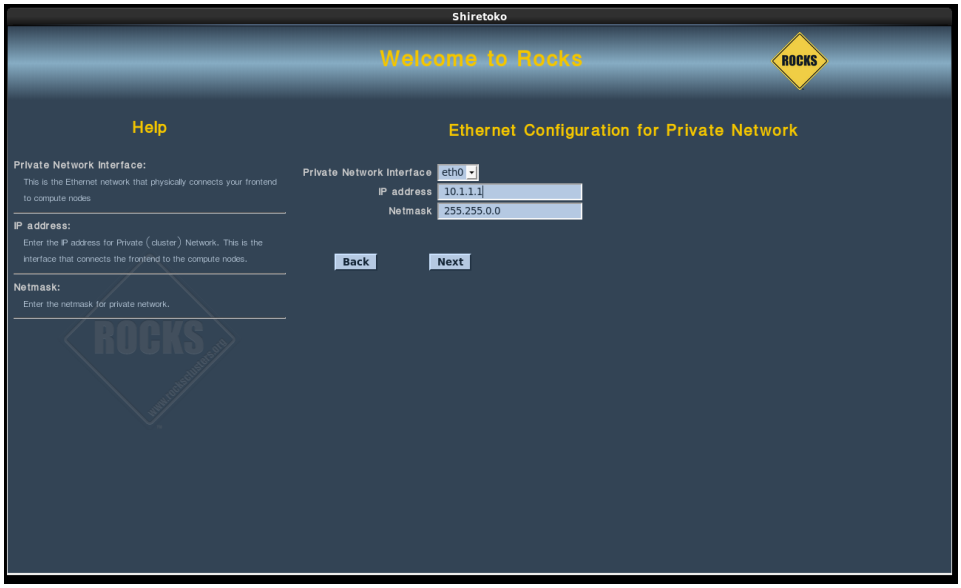


Figura 3.16: Instalación Rocks: Configuración interfaz de red eth0

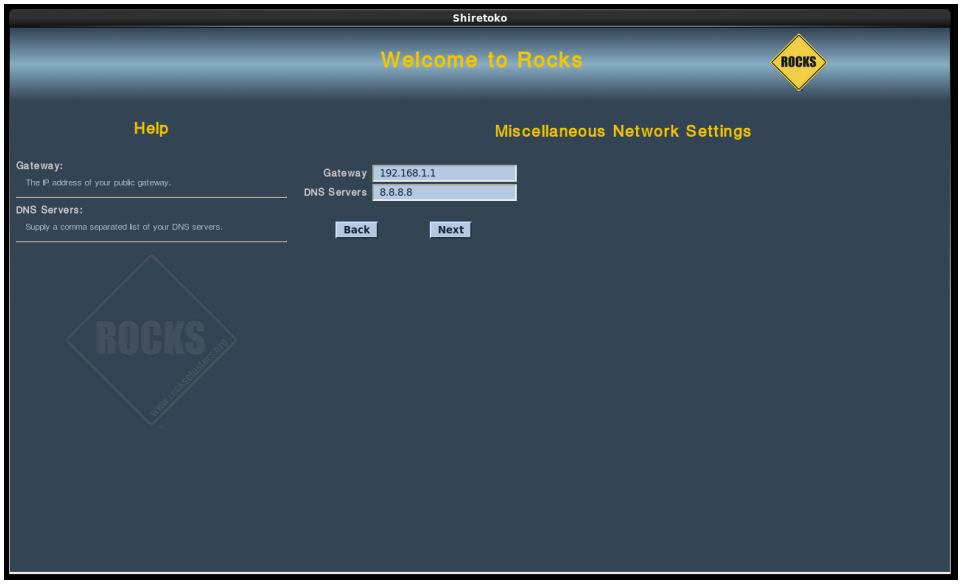


Figura 3.17: Instalación Rocks: Configuración DNS y Gateway

Después de configurar el direccionamiento y los adaptadores de red, se debe de definir una password para el usuario root del sistema, comunmente utilizado en los sistemas GNU/Linux basados en estandares UNIX. Esta cuenta de superusuario es la utilizada para la administración interna del sistema cluster y la aplicación de configuraciones explicitas, instalación de software, ampliación de equipos, etc. Conviene utilizar una contraseña robusta por la repercusión que podría suponer la suplantación del usuario root, pudiendo comprometer el sistema entero, ademas, conviene recordarla pues las gestiones más importantes como la creación de usuarios, actualizaciones, etc, deben de hacerse con elevaciones de privilegios.

Figura 3.18: Instalación Rocks: Definición de contraseña para root

En el siguiente paso, se muestran las opciones que se ofrecen para realizar las particiones de discos, pudiendo optar por manual o no. En una instalación básica se recomienda el modo Automático para que el sistema auto-defina las particiones que considere en función del tamaño del disco, incluida la partición SWAP. En caso de no cumplir el requisito de espacio mínimo, el instalador fallará y se mostrará un error advirtiendo de la necesidad de más espacio en recursos de disco.

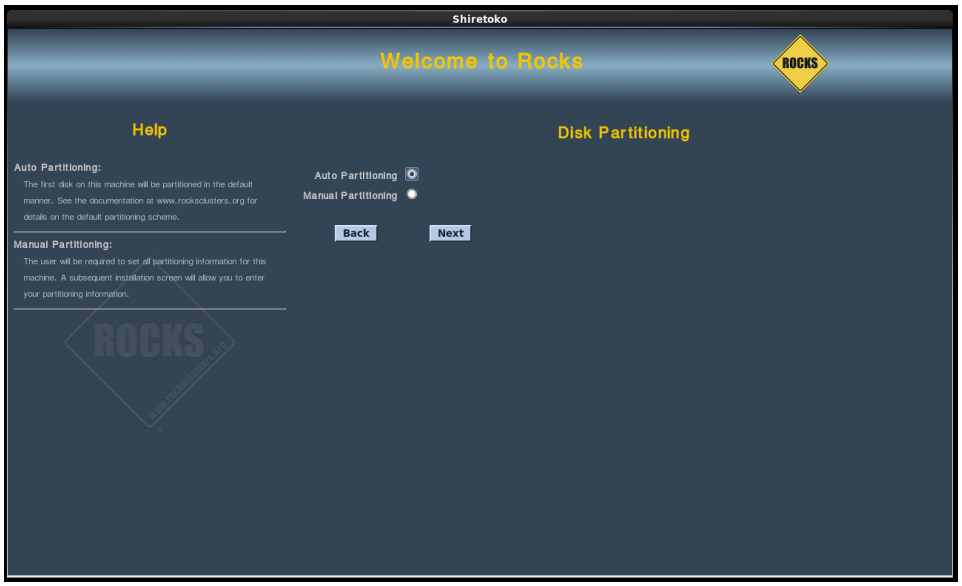


Figura 3.19: Instalación Rocks: Selección modo partición de disco

Las diferentes particiones generadas por el asistente de particionado de discos están descritas a continuación en el siguiente cuadro, esta información ha sido posible obtenerla posteriormente a la instalacion desde el terminal del sistema utilizando el interface Rocks CLI.

DEVICE	MOUNTPOINT	START	SIZE	ID	TYPE	FLAGS	FORMATFLAGS
sda1	/	1049kB	16.8GB	—	ext4	boot	————
sda2	/var	16.8GB	4194MB	—	ext4	———	————
sda3	swap	21.0GB	1049MB	—	linux-swap(v1)	———	————
sda4	————	22.0GB	38.0GB	—	————	———	————
sda5	/state/partition1	22.0GB	38.0GB	—	ext4	———	————

Una vez realizados los pasos detallados anteriormente, empezará la instalación del sistema operativo Rocks en el nodo frontend, con los diferentes Rolls que se hayan seleccionado previamente.

Figura 3.20: Instalación Rocks: Progreso de instalación de sistema Rocks

Cuando finalice el progreso, el sistema estará correctamente instalado y tras un reinicio el nodo frontend arrancará hasta mostrar la pantalla de inicio de sesión. En esta pantalla, se podrá acceder a la interfaz del sistema utilizando las credenciales root definidas anteriormente. A partir de este punto, ya se puede administrar correctamente las configuraciones necesarias para seguir con el proceso de implementación.

3.3.3 Inserción de nodos

El handicap de la utilización de un sistema multiprocesador con memoria distribuida se basa en la utilización de computadores distintos débilmente acoplados, trabajando en conjunto para la resolución de problemas complejos como se ha descrito. La distribución Rocks Cluster tiene la ventaja de que facilita el proceso para añadir los distintos nodos de cómputo y configurarlos de forma sencilla, para que puedan formar parte de la arquitectura del sistema de manera rápida y cómoda utilizando métodos de arranque del Kernel por red.

De forma previa, para utilizar esta ventaja, los equipos que actúen con el rol de nodo de cálculo tienen que habilitar el arranque BIOS por defecto en modo **PXE (Preboot eXecution Environment)**, de esta forma, permitirá el arranque e instalación del sistema a través de la red obteniendo la imagen de instalación correspondiente a cada nodo con la configuración definida en el Frontend. En caso de no disponer de esta opción, se puede arrancar la instalación mediante un formato CD con el kernel del sistema Rocks.

Para incluir un nuevo nodo de cálculo, el sistema operativo Rocks en el nodo Frontend cuenta con un sencillo programa que se ejecuta desde terminal para simplificar el proceso de adición e instalación de nodos en el sistema cluster.

```
[root@cluster ~]# insert-ethers
```

Esto mostrará una pantalla como esta:

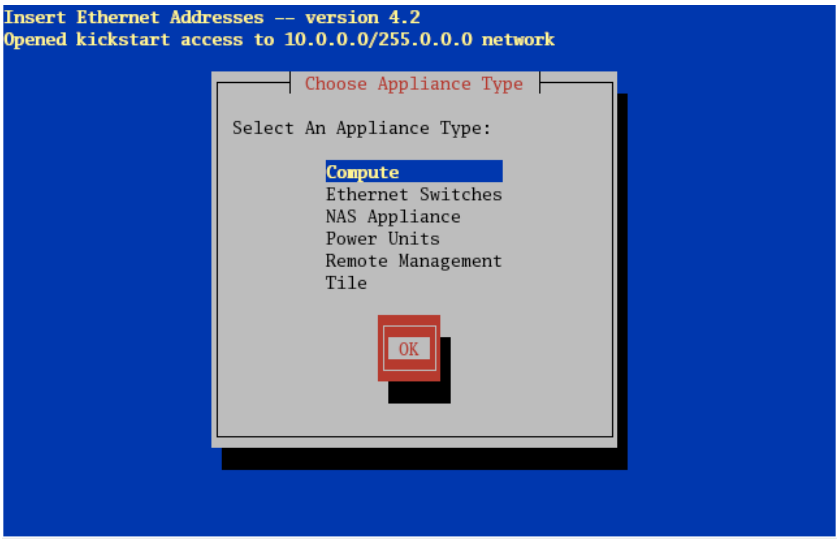


Figura 3.21: Instalación nodos: Selector de recurso a añadir al sistema

Sí el Frontend y los nodos de cálculo están conectados a través de un switch gestionable, se tendría que elegir la opción Ethernet Switches, esto es debido a que el comportamiento por defecto de los switch gestionables con la manipulación de DHCP request es diferente a un switch no gestionable y otra opción podría provocar problemas.

Por defecto se seleccionaría la opción Compute, contando que el switch utilizado no es gestionable, tras esta selección se mostraría una pantalla como la siguiente.

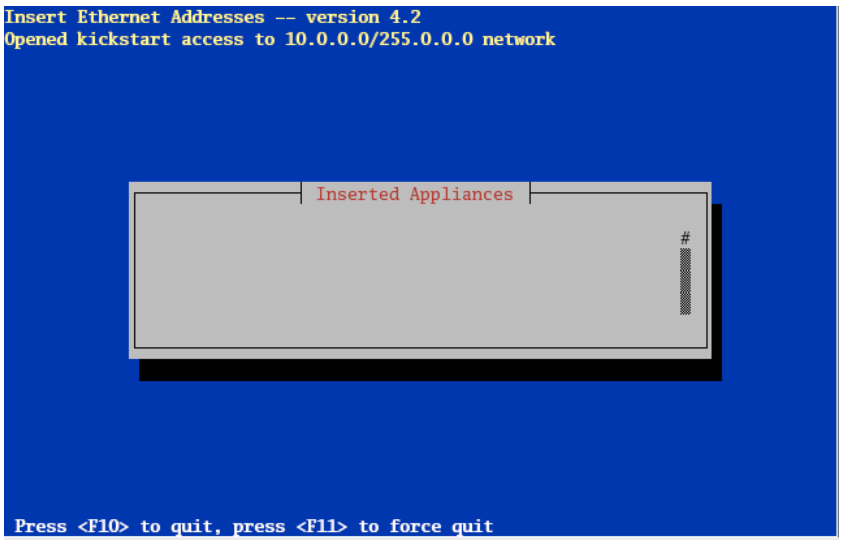


Figura 3.22: Instalación nodos: Pantalla de espera de acks de nodos

Esto indica que el programa de inserción insert-ethers está esperando a nuevos nodos de cálculo.

Es ahora cuando se arrancaría el equipo que se desee instalar como nodo de cálculo, cuando el Frontend recibe la solicitud de la petición DHCP del nodo, aparecerá esta ventana.

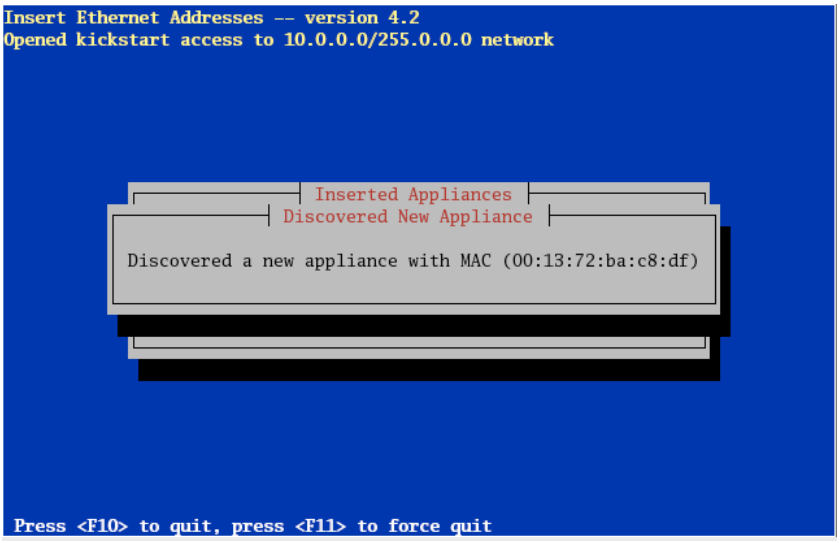


Figura 3.23: Instalación nodos: Descubierta petición de instalación en MAC

Esto indica que se ha recibido una petición del nodo, donde ahora se procederá a insertarlo en la base de datos y actualizar todos los ficheros de configuración para garantizar su correcto funcionamiento.

A continuación se debería de observar la siguiente pantalla, donde el campo "()" indica que el nodo aún no ha hecho la petición del fichero **kickstart**, que es el archivo enviado por red que contendrá el procedimiento para la instalación de un nuevo nodo.

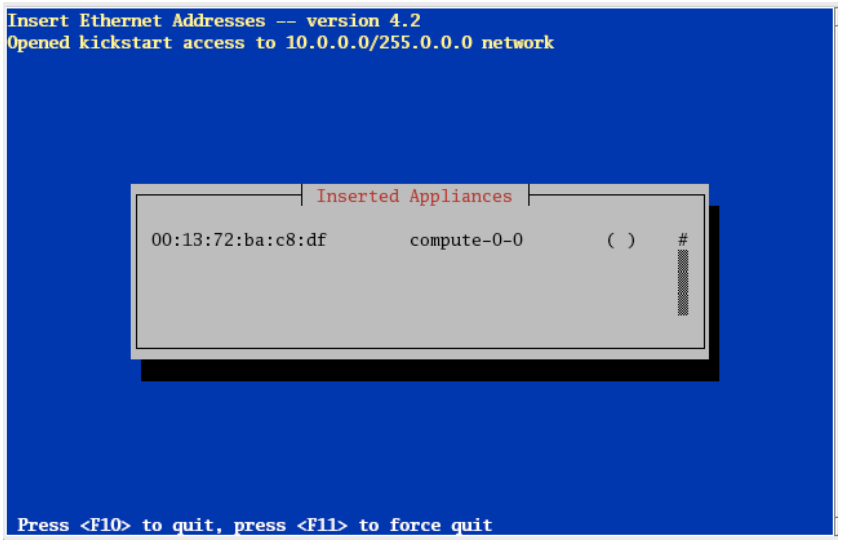


Figura 3.24: Instalación nodos: el nodo compute-0-0 aún no ha recibido el fichero kickstart

Cuando el nodo recibe la petición correctamente y lo confirma, se indicará con (*) que no ha aparecido ningún tipo de error, y se procederá ahora al envío e instalación de la imagen del sistema a través de la red de interconexión con la configuración adecuada para el nodo insertado.

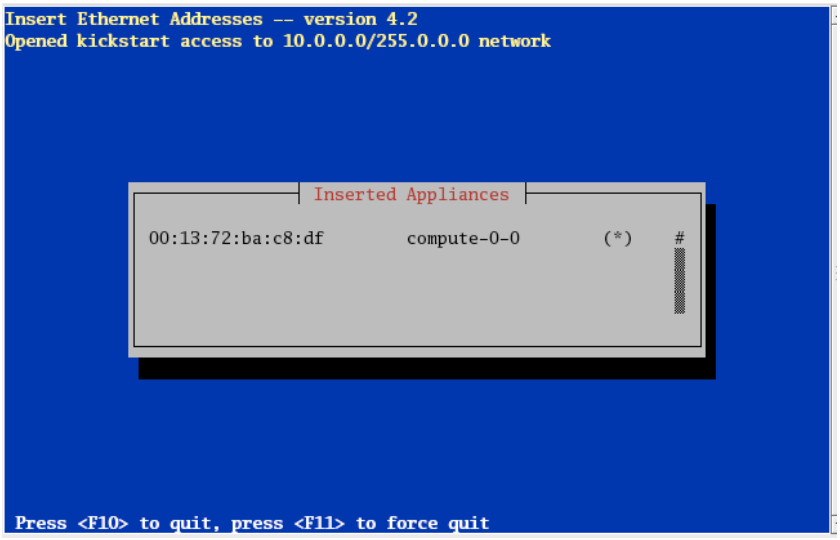


Figura 3.25: Instalación nodos: Símbolo verificación (*) de recepción de kickstart en nodo

Este proceso, se debería de repetir para todos los nodos que se deseen instalar en el cluster, pudiendo iniciar múltiples inserciones de forma simultanea. La instalación se termina con la tecla **F8**. Sí durante la instalación de nodos se quieren clasificar en diferentes ubicaciones, se puede ejecutar el programa **insert-ethers** añadiendo el parámetro –**cabinet=0-N**, indicando en que cabina o armario está instalado el nodo que va a insertarse.

Finalizado el proceso de instalación de los nodos, se puede visualizar una lista de los nodos instalados ejecutando el siguiente comando del kit de herramientas Rocks que se describirá con más detalle posteriormente.

```
[root@cluster ~]# rocks list host
```

```
HOST           MEMBERSHIP CPUS RACK RANK RUNACTION INSTALLACTION
cluster:       Frontend   1    0    0    os        install
compute-0-3:   Compute    1    0    3    os        install
compute-0-2:   Compute    1    0    2    os        install
compute-0-1:   Compute    1    0    1    os        install
compute-0-0:   Compute    1    0    0    os        install
```

Este comando mostrará como resultado todos los nodos insertados en la base de datos del sistema operativo, clasificados según el rol que desempeñan, número de procesadores disponibles, ubicación y acción predeterminada en el arranque.

3.4 Configuración entorno y utilidades

Como se ha comentado anteriormente, una de las bondades de la utilización de la distribución Rocks Cluster es la facilidad que ofrece para la administración del sistema. En este caso, la configuración realizada sobre el nodo Frontend mayoritariamente se realiza a través de una shell, donde con una serie de comandos específicos se puede modificar la configuración general del sistema cluster y sincronizarla con los distintos nodos que lo conformen, además, se cuenta con los programas comunes de GNU/Linux para la administración del sistema.

A continuación se detallarán algunos de los principales recursos y procesos post-instalación del Frontend y nodos, para el correcto funcionamiento del sistema cluster.

3.4.1 Rocks Command Line Interface CLI

A partir de la versión 4.3 de Rocks, se introdujo la función **Rocks Command Line**, para unificar bajo una interface las estructuras de comandos utilizadas para controlar la configuración del sistema y el comportamiento de la arquitectura.

Rocks utiliza una base de datos SQL para guardar información acerca de los nodos, particiones, parámetros de arranque y otra información de toda la instalación actual del sistema. Posteriormente, basándose en la información almacenada en la base de datos, las configuraciones modificadas son actualizadas directamente en los ficheros que almacenan esta información.

La regeneración de los ficheros de configuración ocurre cada vez que un nodo es añadido o borrado del cluster, se modifica algun parámetro de configuración, etc. Un gran porcentaje de los comandos de Rocks se utilizan para manipular la información de configuración almacenada en la base de datos, así de forma general el proceso de cambio de alguna configuración en el sistema se realiza utilizando líneas de comandos especificas para su modifación.

Los comandos de la interface Rocks en consola engloban diferentes operaciones aplicables, pudiendo indicar mediante parámetros las distintas configuraciones aplicables sobre el sistema operativo, Frontend, nodos, o cualquier elemento configurable.

A continuación se muestran algunos de los diferentes parámetros disponibles comúnmente utilizados, para más información acerca de la ejecución de los siguientes comandos se puede utilizar las extensas paginas del manual de uso que se puede obtener con el siguiente comando.

```
[root@cluster ~]# rocks
```

Este comando mostrará una extensa lista de todos los comandos disponibles para realizar modificaciones de la configuración a través de una shell abierta en el nodo de gestión.

Dentro de estas múltiples opciones de configuración, se podrían categorizar en diferentes subtipos ya que los comandos están enfocados a realizar tareas concretas dentro de la configuración del sistema, donde estas modificaciones se aplican directamente sobre los ficheros y actualizados en la base de datos que gestiona la configuración global.

Estos son los distintos subgrupos que se pueden diferenciar:

- **add** - Este comando se utiliza principalmente para añadir elementos al sistema, hosts (elementos máquina) o Rolls (paquetes de software de la distribución Rocks).

  Ejemplos de uso de este comando podrían ser:

- **Añadir un host al cluster:**

```
[root@cluster ~]# rocks add host {host} [cpus=int] [membership=string] [rack=int]
    [rank=int]
```

Añade un host con un nombre simple, se pueden definir parámetros para clasificarlo en un rango dentro de un rack de hosts. A través de argumentos se puede modificar la configuración del host introducido.

[cpus=int] define el nombre de CPUs que serán atribuidas al host. Sí no se determina por defecto será un solo procesador.

[membership=string] Determina el grupo donde pertenece el host.

[rack=int] Determina el número de rack donde el host está ubicado.

[rank=int] Determina la posición del host dentro del rack donde está ubicado.

Un posible funcionamiento sería:

```
[root@cluster ~]# rocks add host frontend rack=0 rank=0 membership=Compute
```

Donde el comando añadiría un host definiendo 1 cpu, miembro del grupo de computación y ubicado en el rack 0 en primera posición.

- **create** - Este comando se utiliza para empaquetar el software desde diferentes orígenes y facilitar en el futuro el despliegue sobre otro sistema o para actualizaciones posteriores. Permite crear tres variantes, distros, packages o rolls.

  Una posibilidad de uso de este comando podría ser:

  - **Creacion de un Roll release de software:**

  Este comando crea un Roll de software a partir de un fichero XML, o un MetaRoll a partir de una lista de isos. El argumento utilizado es: **roll**. En este argumento se pasa o bien una lista de ISOS para realizar un merge en una Roll sencilla, o el nombre de un archivo XML generado donde figura la descripción de todos los orígenes disponibles.

  Un ejemplo de uso sería:

```
[root@cluster ~]# rocks create roll roll-base.xml
```

  Esto generaría una Roll base generada a partir del fichero roll-base.xml.

```
[root@cluster ~]# rocks create roll {roll...}
```

- **list** - Este comando permite obtener información relevante tanto del cluster como de los host introducidos en el mismo, donde los parámetros definen que características se quieren visualizar en la salida del comando, obteniendo información de las configuraciones aplicadas.

  También se pueden encontrar modificadores del comando **rocks list** para poder listar características principales del cluster, donde se podrían destacar distintas opciones posibles.

  La sintaxis del comando es la siguiente:

```
[root@cluster ~]# rocks list [function] [param] [host]
```

El parámetro **[function]** define la opción que se desea obtener, donde se puede combinar con los parámetros **[param]** que definen las características que se quieren comprobar de un host, y donde el argumento **[host]** define el identificador del host sobre el que aplicar la consulta total.

Estas son algunas de las opciones disponibles dentro del comando rocks list.

list appliance [attr, route, xml][appliance]
list attr
list bootaction
list distribution [distribution]
list firewall [category=index] [maxwidth=integer]
list help [subdir=string]
list host [alias, appliance, attr, boot, firewall, graph, installfile, interface, key, macs, membership, partition, profile, roll, route, sec_attr, xml] [host]
list license
list membership [membership]
list network [network]
list node xml [attrs=string] [basedir=string] [eval=bool] [gen=string] [missing-check=bool] [roll=string]
list os [attr, route] [os]
list roll [command] [roll]
list route
list sec_attr
list var [appliance=string] [component=string] [service=string]

A continuación se pueden observar algunas de las características obtenidas de la ejecución del comando rocks list sobre el cluster configurado.

- **Listar acción de arranque por red de un nodo**

```
[root@cluster ~]# rocks list bootaction
ACTION          KERNEL                  RAMDISK             ARGS
install:        vmlinuz-6.1-i386        initrd.img-6.1-i386 ks
ramdisk_size=150000 lang= devfs=nomount pxe kssendmac selinux=0 noipv6 ksdevice=
    bootif
install headless: vmlinuz-6.1-i386      initrd.img-6.1-i386 ks
ramdisk_size=150000 lang= devfs=nomount pxe kssendmac selinux=0 noipv6 headless
    vnc ksdevice=bootif

memtest:        kernel memtest          ---------------------
os:             localboot 0             ---------------------
pxeflash:       kernel memdisk bigraw   pxeflash.img        keeppxe
rescue:         vmlinuz-6.1-i386        initrd.img-6.1-i386 ks

ramdisk_size=150000 lang= devfs=nomount pxe kssendmac selinux=0 noipv6 rescue
    ksdevice=bootif
```

- **Listar rolls instaladas en el sistema cluster**

```
[root@cluster ~]# rocks list roll
NAME           VERSION     ARCH ENABLED
sge:           6.1         i386 yes
zfs-linux:     0.6.0.rc12  i386 yes
os:            6.1         i386 yes
perl:          6.1         i386 yes
ganglia:       6.1         i386 yes
service-pack:  6.1         i386 yes
area51:        6.1         i386 yes
base:          6.1         i386 yes
web-server:    6.1         i386 yes
python:        6.1         i386 yes
condor:        6.1         i386 yes
hpc:           6.1         i386 yes
bio:           6.1         i386 yes
java:          6.1         i386 yes
kernel:        6.1         i386 yes
```

- **Listar configuración de red actual en el cluster**

```
[root@cluster ~]# rocks list network
NETWORK  SUBNET       NETMASK         MTU   DNSZONE     SERVEDNS
private: 10.1.0.0     255.255.0.0     1500  local       True
public:  192.168.1.0  255.255.255.0   1500  torrex.org  False
```

- **Listar arranque por defecto definido en los nodos**

```
[root@cluster ~]# rocks list host boot
HOST           ACTION
cluster:       ------
compute-0-3:   os
compute-0-2:   os
compute-0-1:   os
compute-0-4:   os
```

- **Listar particiones de disco del sistema**

```
[root@cluster ~]# rocks list host partition
HOST           DEVICE MOUNTPOINT          START   SIZE    ID TYPE             FLAGS
    FORMATFLAGS
cluster:       sda1   /                   1049kB  159GB   -- ext4             boot
cluster:       sda2   swap                159GB   1048MB  -- linux-swap(v1)
compute-0-3:   sda1   /                   1049kB  16.8GB  -- ext4             boot
compute-0-3:   sda2   /var                16.8GB  4194MB  -- ext4
compute-0-3:   sda3   swap                21.0GB  1049MB  -- linux-swap(v1)
compute-0-3:   sda4   ------------------  22.0GB  18.0GB  -- ----------------
compute-0-3:   sda5   /state/partition1   22.0GB  18.0GB  -- ext4
compute-0-2:   sda1   /                   1049kB  16.8GB  -- ext4             boot
compute-0-2:   sda2   /var                16.8GB  4194MB  -- ext4
compute-0-2:   sda3   swap                21.0GB  1049MB  -- linux-swap(v1)
compute-0-2:   sda4   ------------------  22.0GB  18.0GB  -- ----------------
compute-0-2:   sda5   /state/partition1   22.0GB  18.0GB  -- ext4
compute-0-1:   sda1   /                   1049kB  16.8GB  -- ext4             boot
compute-0-1:   sda2   /var                16.8GB  4194MB  -- ext4
compute-0-1:   sda3   swap                21.0GB  1049MB  -- linux-swap(v1)
compute-0-1:   sda4   ------------------  22.0GB  38.0GB  -- ----------------
compute-0-1:   sda5   /state/partition1   22.0GB  38.0GB  -- ext4
compute-0-4:   sda1   /                   1049kB  16.8GB  -- ext4             boot
compute-0-4:   sda2   /var                16.8GB  4194MB  -- ext4
compute-0-4:   sda3   swap                21.0GB  1049MB  -- linux-swap(v1)
compute-0-4:   sda4   ------------------  22.0GB  38.0GB  -- ----------------
compute-0-4:   sda5   /state/partition1   22.0GB  38.0GB  -- ext4
```

- **Listar las interfaces de red del sistema**

```
[root@cluster ~]# rocks list host interface
HOST           SUBNET   IFACE MAC                IP              NETMASK         MODULE
    NAME       VLAN OPTIONS CHANNEL
cluster:       private  eth0  00:XX:85:1E:35:D7  10.1.1.1        255.255.0.0
cluster:       public   eth1  00:XX:XX:C1:AF:19  192.168.1.69    255.255.255.0
compute-0-3:   private  eth0  00:XX:XX:5d:81:10  10.1.255.251    255.255.0.0
compute-0-2:   private  eth0  00:XX:XX:c1:b0:fc  10.1.255.252    255.255.0.0
compute-0-1:   private  eth0  00:XX:XX:06:30:1d  10.1.255.253    255.255.0.0
compute-0-4:   private  eth0  00:XX:XX:05:ff:d7  10.1.255.254    255.255.0.0
```

- **remove** - Este comando se utiliza para eliminar elementos del sistema o su configuración desde la shell. Su utilización está principalmente orientada a eliminar hosts o componentes del software del cluster.

  El comando mantiene la siguiente sintaxis:

```
[root@cluster ~]# rocks remove [function] (args)
```

  A continuación se listan algunas de sus opciones disponibles.

  remove appliance [attr][route] name
  remove attr [attr=string]
  remove bootaction [action=string]
  remove distribution
  remove firewall [category=index] rulename
  remove host alias [attr] [boot] [bootflags][interface][key][partition][roll][route][sec_attr] host ...
  remove network network...
  remove os [attr] [route] os ...
  remove roll... [arch=string] [version=string]
  remove route address [address=string]
  remove sec_attr attr

  Ejemplos de uso del siguiente comando podrían ser:

  - **Remover interface de red de un nodo**

```
[root@cluster ~]# rocks remove host interface compute-0-0 eth1
```

  - **Eliminar partición de un nodo**

```
[root@cluster ~]# rocks remove host partition compute-0-0 partition=/export
```

- **run** - Este comando principalmente se utiliza para ejecutar en un host un programa o acción desde una shell remota. Esto es útil para realizar mantenimiento sin tener que acceder al host físicamente mediante elementos de entrada/salida, permitiendo acciones a través de túnel SSH.

  La sintaxis del comando sería la siguiente:

```
[root@cluster ~]# rocks run host [host] [command] {args}
```

  Un ejemplo de uso del comando podría ser:

  - **Apagado remoto de un host del sistema**

```
[root@cluster ~]# rocks run host compute-0-0 poweroff
```

- **set** - Este comando se utiliza para definir la mayoría de variables que afectan a la configuración del sistema. Esta opción puede englobar desde configuraciones de red hasta atributos propios de un host que se deseen configurar.

  La sintaxis del comando es la siguiente:

```
[root@cluster ~]# rocks set [function] (args)
```

  A continuación se listan algunas de sus opciones disponibles.

  set appliance attr appliance attr value
  set attr attr value
  set host attr host attr value
  set host [boot][bootflags][comment] [cpus][installaction][interface][membership][name][rack][ran
  host
  set network [mtu][netmask][servedns][subnet][zone][netmask] networkmtu]
  set os attr os attr value
  set sec_attr attr
  set var service component value

  Algunos ejemplos de uso podrían ser:

  - **Definir número de CPUS para un host**

```
[root@cluster ~]# rocks set host cpus compute-0-0 2
```

  - **Definir una IP en una interface de red**

```
[root@cluster ~]# rocks set host interface ip compute-0-0 eth1 192.168.0.10
```

  - **Definir puerta de enlace para un host**

```
[root@cluster ~]# rocks set host interface gateway compute-0-0 iface=eth1 gateway
    =192.168.0.1
```

- **sync** - Este comando se utiliza para realizar la sincronización de la configuración que se defina a través de parámetros. Se puede utilizar para realizar una sincronización masiva en todos los host activos, donde dentro de sus opciones se pueden encontrar parámetros de sincronización para: configuraciones generales, DNS, firewall, network, keys de certificado o usuarios, etc.

  La sintaxis del comando es la siguiente:

```
[root@cluster ~]# rocks sync [users] [config]
```

  La sincronización de la **configuración** obtendría un rebuild de los ficheros de configuración y provocaría reinicio de los servicios activos.

  La sincronización de los **usuarios** actualizaría todos los archivos relacionados con administración de usuarios, como las contraseñas habilitadas en todos los hosts conocidos o las huellas generadas de autenticación.

- **update** - Este comando de configuración se utiliza para realizar la actualización del software instalado en la distribución.

3.4.2 Creación de usuarios del sistema y sincronización

Al disponer de una distribución GNU/Linux, la gestión de usuarios y permisos es una parte básica del sistema operativo, ofreciendo la oportunidad de administrar una parte importante en la seguridad y privacidad sobre las funcionalidades y datos existentes.

Por defecto, la instalación de la distribución Rocks arranca con el usuario root del sistema para proporcionar todos los permisos necesarios para una correcta configuración previa, además, este está sincronizado en todos los nodos que se han instalado posteriormente en el cluster.

En la práctica, la utilización del usuario root puede estar contraindicada, pues una falta de experiencia o un desliz en la configuración puede comprometer la estabilidad o la seguridad de la infraestructura.

Para realizar las distintas pruebas en el sistema cluster del proyecto actual, se creará un usuario alternativo, pues también algunos programas no permiten su ejecución en modo superusuario (root), por ejemplo mpirun.

El patrón de creación de usuarios se rige mediante estándares UNIX, ejecutando desde el shell en el Frontend los siguientes comandos.

```
[root@cluster ~]# useradd zxcv90
[root@cluster ~]# passwd zxcv90
[root@cluster ~]# (Introduccion de password deseado)
```

Esto resulta básico en cualquier sistema GNU/Linux, pero el detalle en esta distribución Rocks radica en que este usuario solo se ha creado a nivel local en el Frontend, y es necesario sincronizarlo a los diferentes nodos.

Para poder realizar la sincronización es necesaria la ejecución de la utilidad Rocks descrita anteriormente con los siguientes parámetros:

```
[root@cluster ~]# rocks sync users
```

Esto se encargará de sincronizar toda la configuración de usuarios en todos los nodos conectados al Frontend.

3.4.3 Modificación arranque de nodos

A través de la herramienta **rocks list bootaction**, se puede obtener el patrón de arranque que seguirá el hilo de ejecución cuando un nodo se tenga que insertar en la configuración del sistema.

La ejecución de este comando arroja una salida como esta, donde esta información es modificable, por lo que en todo momento se puede gestionar que kernel e imagen de sistema se está volcando a la memoria de los nodos de cómputo en caso de actualización.

```
ACTION              KERNEL                  RAMDISK                 ARGS
install:            vmlinuz-6.1-i386        initrd.img-6.1-i386 ks ramdisk_size=150000 lang
    = devfs=nomount pxe kssendmac selinux=0 noipv6 ksdevice=bootif
install headless: vmlinuz-6.1-i386          initrd.img-6.1-i386 ks ramdisk_size=150000 lang
    = devfs=nomount pxe kssendmac selinux=0 noipv6 headless vnc ksdevice=bootif
memtest:            kernel memtest          ------------------------
    ------------------------------------------------
os:                 localboot 0             ------------------------
    ------------------------------------------------
pxeflash:           kernel memdisk bigraw pxeflash.img          keeppxe
```

```
rescue:          vmlinuz-6.1-i386      initrd.img-6.1-i386 ks ramdisk_size=150000 lang
    = devfs=nomount pxe kssendmac selinux=0 noipv6 rescue ksdevice=bootif
```

3.4.4 Distribución ficheros en nodos: /share/apps

La gestión de parte del sistema de ficheros mediante NFS (Network File System) permite compartir cualquier archivo o programa que se desee a través de la red, para dejarlo accesible desde todos los nodos del cluster.

El directorio **/share/apps** es sincronizado entre el Frontend y el resto de nodos, estará disponible de manera inmediata en todos, por lo que es un buen recurso para depositar ficheros que requieran de su disponibilidad en todas las máquinas.

3.4.5 Sun Grid Engine: Sistema de colas de trabajos

Sun Grid Engine es un sistema de colas de trabajos desarrollado como software de código abierto por Sun Microsystems, aportando su funcionalidad a la distribución Rocks a través del Roll SGE para lanzar trabajos programados en todo el cluster, para facilitar la gestión de recursos del sistema o planificación de ejecuciones.

SGE se encarga de diversas funciones en la gestión de tareas en el espacio de usuario, ya sean estas secuenciales, paralelas o interactivas.

Las tareas a ejecutar, son enviadas a Sun Grid Engine utilizando scripts de shell, donde se simplifica el proceso para automatizar pruebas recurrentes o con modificación de parámetros constantes. Un ejemplo sería el siguiente script, sge-qsub-test.sh que se puede utilizar de forma genérica para testear el Sun Grid Engine, facilitado como plantilla.

```
#!/bin/bash
#$ -S /bin/bash
#
# set the P4_GLOBMEMSIZE
#$ -v P4_GLOBMEMSIZE=10000000
#
# Set the Parallel Environment and number of procs.
#$ -pe mpi 2

# Where we will make our temporary directory.
BASE="/tmp"

#
# make a temporary key
#
export KEYDIR=`mktemp -d $BASE/keys.XXXXXX`

#
# Make a temporary password.
# Makepasswd is quieter, and presumably more efficient.
# We must use the -s 0 flag to make sure the password contains no quotes.
#
if [ -x `which mkpasswd` ]; then
export PASSWD=`mkpasswd -l 32 -s 0`
else
export PASSWD=`dd if=/dev/urandom bs=512 count=100 | md5sum | gawk '{print $1}'`
fi

/usr/bin/ssh-keygen -t rsa -f $KEYDIR/tmpid -N "$PASSWD"

cat $KEYDIR/tmpid.pub >> $HOME/.ssh/authorized_keys2
```

```
#
# make a script that will run under its own ssh-agent
#
cat > $KEYDIR/launch-script <<"EOF"
#!/bin/bash
expect -c 'spawn /usr/bin/ssh-add $env(KEYDIR)/tmpid' -c \
'expect "Enter passphrase for $env(KEYDIR)/tmpid" \
{ send "$env(PASSWD)\n" }' -c 'expect "Identity"'

echo

#
# Put your Job commands here.
#
#-----------------------------------------------

/opt/mpich/gnu/bin/mpirun -np $NSLOTS -machinefile $TMP/machines \
/opt/hpl/gnu/bin/xhpl

#-----------------------------------------------
EOF

chmod u+x $KEYDIR/launch-script

#
# start a new ssh-agent from scratch -- make it forget previous ssh-agent
# connections
#
unset SSH_AGENT_PID
unset SSH_AUTH_SOCK
/usr/bin/ssh-agent $KEYDIR/launch-script

#
# cleanup
#
grep -v "`cat $KEYDIR/tmpid.pub`" $HOME/.ssh/authorized_keys2 \
> $KEYDIR/authorized_keys2
mv $KEYDIR/authorized_keys2 $HOME/.ssh/authorized_keys2
chmod 644 $HOME/.ssh/authorized_keys2

rm -rf $KEYDIR
```

El script internamente realiza el encolado en SGE para lanzar una ejecución de un programa MPI con $NSLOTS$ procesos. Este script está diferenciado en diferentes secciones, donde entre ellas establece una clave ssh temporal usada por el usuario para realizar la ejecución del programa mediante mpirun.

Para realizar el envío del job a SGE se deberían de ejecutar los siguientes comandos en el shell del sistema Frontend:

```
[root@cluster GridEngine]# qsub sge-qsub-test.sh
Your job 14 ("sge-qsub-test.sh") has been submitted
```

Cuando la tarea se ha ejecutado, se puede comprobar el estado de la cola mediante el siguiente comando:

```
[root@cluster ~]# qstat -f
```

A continuación se muestran las salidas ofrecidas por la ejecución del comando anterior, en el primer caso se muestra la cola de SGE con una tare pendiente, en el segundo caso se muestra un a tarea en ejecución dentro de la cola en dos nodos.

```
[root@cluster ~]# qstat -f
queuename                      qtype resv/used/tot. load_avg arch          states
---------------------------------------------------------------------------------
all.q@compute-0-0.local        BIP   0/0/1          0.11     linux-x86
---------------------------------------------------------------------------------
all.q@compute-0-1.local        BIP   0/0/1          0.10     linux-x86
---------------------------------------------------------------------------------
all.q@compute-0-2.local        BIP   0/0/1          -NA-     linux-x86     au
---------------------------------------------------------------------------------
all.q@compute-0-3.local        BIP   0/0/2          -NA-     linux-x86     au

############################################################################
- PENDING JOBS - PENDING JOBS - PENDING JOBS - PENDING JOBS - PENDING JOBS
############################################################################
14 0.00000 sge-qsub-t root         qw    08/24/2016 18:21:55     2
```

```
[root@cluster ~]# qstat -f
queuename                      qtype resv/used/tot. load_avg arch          states
---------------------------------------------------------------------------------
all.q@compute-0-0.local        BIP   0/1/1          0.11     linux-x86
14 0.55500 sge-qsub-t root         r     08/24/2016 18:22:00     1
---------------------------------------------------------------------------------
all.q@compute-0-1.local        BIP   0/1/1          0.10     linux-x86
14 0.55500 sge-qsub-t root         r     08/24/2016 18:22:00     1
---------------------------------------------------------------------------------
all.q@compute-0-2.local        BIP   0/0/1          -NA-     linux-x86     au
---------------------------------------------------------------------------------
all.q@compute-0-3.local        BIP   0/0/2          -NA-     linux-x86     au
```

Los lanzamientos de tareas a la cola de ejecución, son registrados por Sun Grid Engine almacenando información post-ejecución en una serie de archivos generados cuando se ejecuta una tarea.

Los ficheros de salida generados tienen una descripción nominal similar a:

sge-qsub-test.sh.o<job id>(stdout messages)

sge-qsub-test.sh.e<job id>(stderr messages)

La utilización de esta herramienta es verdaderamente potente para facilitar la ejecución de programas donde se necesiten obtener resultados diferentes dependiendo de varios parámetros o configuraciones. Resulta muy sencillo programar scripts de shell propios que se adecuen a las necesidades de una determinada ejecución para implementarlos y simplificar su lanzamiento, ahorrando un tiempo valioso en el tiempo dedicado a lanzar múltiples ejecuciones diferentes sobre el cluster.

3.5 Test del entorno: Comunicación y ejecución remota

Con el sistema configurado, para verificar el correcto funcionamiento del sistema se puede proceder a la ejecución de algunos sencillos test para comprobar que todo está operativo y configurado correctamente.

Un test sencillo en el entorno del proyecto donde existe una cantidad pequeña de ordenadores, es comprobar la conexión entre ellos, pudiendo ejecutar por ejemplo PING desde el Frontend para verificar que los hosts responden adecuadamente. El ping se puede realizar al nombre del host obtenido anteriormente con la ejecución de **rocks list host**. En caso de querer comprobar

arquitecturas con mayor envergadura en el número de nodos, se debería de recurrir a sistemas de monitorización completos como Ganglia, descrito en las próximas secciones.

Con la realización de un ping sencillo a un host del sistema:

```
[root@cluster ~]# ping compute-0-0
```

Se puede comprobar la respuesta por parte de los diferentes host instalados en el cluster:

```
PING compute-0-0.local (10.1.255.254) 56(84) bytes of data.
64 bytes from compute-0-0.local (10.1.255.254): icmp_seq=1 ttl=64 time=0.254 ms
64 bytes from compute-0-0.local (10.1.255.254): icmp_seq=2 ttl=64 time=0.211 ms
64 bytes from compute-0-0.local (10.1.255.254): icmp_seq=3 ttl=64 time=0.193 ms
64 bytes from compute-0-0.local (10.1.255.254): icmp_seq=4 ttl=64 time=0.200 ms
^C
--- compute-0-0.local ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3266ms
rtt min/avg/max/mdev = 0.193/0.214/0.254/0.027 ms
```

Esta información verifica que los diferentes nodos están accesibles a través de la red de interconexión, por lo que se puede comunicar con ellos y validar que la estructura de red está recibiendo comunicación desde el nodo Frontend.

A continuación, otro paso interesante a verificar es que se puedan ejecutar comandos lanzados desde el nodo central Frontend a los diferentes nodos de la red de computación, de manera sencilla, sin la utilización de SGE.

Esta condición se puede verificar con la ayuda de programas en ejecución remota a través de utilidades en el Rocks CLI facilitado en la distribución.

```
run host [host]... {command} [collate=string] [command=string] [delay=string] [managed=
    boolean] [num-threads=string] [stats=string] [timeout=string] [x11=boolean]
```

El comando **rocks run host** permite la ejecución de programas en los nodos especificados mediante parámetros. Se podría verificar que funciona la ejecución remota, ejecutando el programa **ps** en en el nodo de cálculo 1 **compute-0-0**, por ejemplo.

```
[root@cluster ~]# rocks run host compute-0-0 ps
```

Sí se ejecuta correctamente, la salida del comando mostrará los procesos activos en el host especificado, esto puede resultar útil en la monitorización de procesos activos dentro de cada nodo, pudiendo habilitar ventanas de terminal independiente para visualizar los procesos activos en diferentes nodos.

La salida del comando anterior podría ser como la siguiente, obviando algunos de los procesos activos que pueda haber ya que aproximadamente en estado idle de un host se pueden ejecutar cientos de procesos.

```
PID TTY          TIME CMD
1 ?        00:00:01 init
2 ?        00:00:00 kthreadd
3 ?        00:00:00 migration/0
4 ?        00:00:00 ksoftirqd/0
5 ?        00:00:00 migration/0
6 ?        00:00:00 watchdog/0
7 ?        00:00:00 events/0
8 ?        00:00:00 cgroup
9 ?        00:00:00 khelper
10 ?       00:00:00 netns
.....................................
```

```
1201 ?        00:00:00 hald-runner
1229 ?        00:00:00 hald-addon-inpu
1319 ?        00:00:00 snmpd
1332 ?        00:00:00 sshd
1430 ?        00:00:00 master
1440 ?        00:00:00 crond
1464 ?        00:00:00 atd
1577 ?        00:00:00 automount
1824 ?        00:00:00 sshd
1828 ?        00:00:00 ps
```

3.6 Monitorización del cluster

3.6.1 Correo

El sistema Rocks dispone de un cliente tradicional de correo UNIX, donde en la variable **/var/spool/mail/$USER** se puede encontrar la bandeja de entrada de correo recibido.

```
[root@cluster cpu]# Tiene correo nuevo en /var/spool/mail/root
```

El correo almacenado en esta variable permanece hasta que el usuario decida no archivarlos, en la distribución Rocks Cluster, podemos encontrar en el correo los logs generados principalmente con Tripwire, explicado posteriormente, así como avisos o información importante.

Para realizar la lectura del correo electrónico, se invoca al programa **mail**, obteniendo una lista del correo pendiente y esperando el programa a que el usuario lo lea, responda, borre o almacene para más tarde.

```
[root@cluster ~]# mail
Heirloom Mail version 12.4 7/29/08.  Type ? for help.
"/var/spool/mail/root": 77 messages 77 new
>N  1 Cron Daemon            Tue Jul 12 17:01  25/924   "Cron <root@cluster> r"
N  2 logwatch@cluster.tor  Tue Jul 12 17:11  39/1341  "Logwatch for cluster."
N  3 Mail Delivery System  Tue Jul 12 17:16 262/10473 "Undelivered Mail Retu"
N  4 logwatch@cluster.tor  Wed Jul 13 17:14 198/7538  "Logwatch for cluster."
N  5 Mail Delivery System  Wed Jul 13 17:19 805/31653 "Undelivered Mail Retu"
N  6 logwatch@cluster.tor  Mon Jul 18 17:25  39/1341  "Logwatch for cluster."
N  7 Mail Delivery System  Mon Jul 18 17:26 906/36703 "Undelivered Mail Retu"
N  8 logwatch@local        Sat Jul 23 13:11  44/1600  "Logwatch for compute-"
N  9 logwatch@local        Sat Jul 23 13:26  44/1600  "Logwatch for compute-"
N 10 logwatch@local        Sat Jul 23 13:35  44/1600  "Logwatch for compute-"
N 11 logwatch@cluster.tor  Sat Jul 23 13:44  39/1341  "Logwatch for cluster."
N 12 logwatch@local        Sat Jul 23 13:44  44/1600  "Logwatch for compute-"
N 13 Mail Delivery System  Sat Jul 23 13:45 875/33870 "Undelivered Mail Retu"
N 14 logwatch@cluster.tor  Mon Jul 25 19:06  39/1341  "Logwatch for cluster."
N 15 Mail Delivery System  Mon Jul 25 19:07 1024/41957 "Undelivered Mail Ret"
N 16 logwatch@local        Mon Jul 25 19:08  44/1600  "Logwatch for compute-"
N 17 logwatch@local        Mon Jul 25 19:10  44/1600  "Logwatch for compute-"
N 18 logwatch@local        Mon Jul 25 19:24  44/1600  "Logwatch for compute-"
N 19 logwatch@local        Mon Jul 25 19:36  44/1600  "Logwatch for compute-"
N 20 logwatch@cluster.tor  Thu Jul 28 18:12  39/1341  "Logwatch for cluster."
&
```

Como se puede observar, mail muestra los mensajes pendientes por responder, con su usuario de origen, la fecha y el tema, con el cursor virtual (>) ubicado sobre el primero indicando sobre cual ejecutará una orden de lectura, respuesta o borrado, además de un prompt con el carácter **&** a la espera de recepción de órdenes sobre qué hacer con dicho correo.

A la mayoría de estas opciones listadas a continuación, se les puede pasar un número de mensaje (desde 1 hasta n) para indicar sobre qué mensaje realizar la acción (ejemplo 'd 3' o 'delete 3').

Las ordenes disponibles son:

- **t** listar mensaje
- **n** ir al mensaje especificado y listarlo.
- **e** editar mensaje
- **f** ver cabeceras del mensaje
- **d** borrar mensaje
- **s** añadir mensajes a un fichero
- **u** recuperar mensajes borrados
- **R** Responder a los remitentes del mensaje
- **r** Responder al remitente y a todos los destinatarios.
- **pre** hacer ir los mensajes de nuevo a /usr/spool/mail
- **m** enviar mensaje a los usuarios especificados.
- **q** salir grabando mensajes en mbox
- **h** mostrar cabeceras activas.
- **!** permite ejecutar una shell o comandos de shell

Algunos de los mensajes recibidos en el buzón de correo pueden originarse de diferentes eventos generados por aplicaciones de monitorización, donde las más destacadas serian **Logwatch** y **Tripwire**.

A continuación se pueden observar fragmentos de un correo con el registro de logs generados por Logwatch, con eventos significativos para la administración del sistema.

```
Message  2:
From root@cluster.torrex.org  Tue Jul 12 17:11:05 2016
Return-Path: <root@cluster.torrex.org>
X-Original-To: root
Delivered-To: root@cluster.torrex.org
To: root@cluster.torrex.org
From: logwatch@cluster.torrex.org
Subject: Logwatch for cluster.torrex.org (Linux)
Content-Type: text/plain; charset="iso-8859-1"
Date: Tue, 12 Jul 2016 17:11:04 +0200 (CEST)
Status: R

################### Logwatch 7.3.6 (05/19/07) ####################
Processing Initiated: Tue Jul 12 17:11:04 2016
Date Range Processed: yesterday
( 2016-Jul-11 )
Period is day.
Detail Level of Output: 0
Type of Output: unformatted
Logfiles for Host: cluster.torrex.org
##################################################################
--------------------- Disk Space Begin ----------------------
```

```
Filesystem            Size  Used Avail Use% Mounted on
/dev/sda1             146G  7.7G  131G   6% /

------------------------ Disk Space End ------------------------
######################## Logwatch End #########################
 ------------------------ SSHD Begin ------------------------
 SSHD Started: 3 Time(s)

 Users logging in through sshd:
 root:
 10.1.1.1 (cluster.local): 3 times

 Received disconnect:
 11: disconnected by user : 3 Time(s)

 ------------------------ SSHD End ------------------------
```

3.6.2 Ganglia

Ganglia es un sistema de monitorización para sistemas de alto rendimiento como clusters o mallas. Está basado en un diseño jerárquico orientado para clusters, donde utiliza tecnologías como XML para la representación de datos, XDR para el transporte de datos y RRDtool para el almacenamiento de datos y visualización. Utiliza estructuras de datos y algoritmos propios para almacenar el más mínimo cambio en las métricas supervisadas de los nodos en escenarios de alta concurrencia. La implementación es robusta, portada a un gran extenso set de sistemas operativos y arquitecturas de procesadores, es por ello que es ampliamente utilizada en centenares de clusters alrededor del mundo. Entre algunos de sus usos, se podría destacar su utilización para supervisar enlaces de clusters entre campus universitarios con una escalabilidad capacitada para manejar aproximadamente 2000 nodos por cluster.

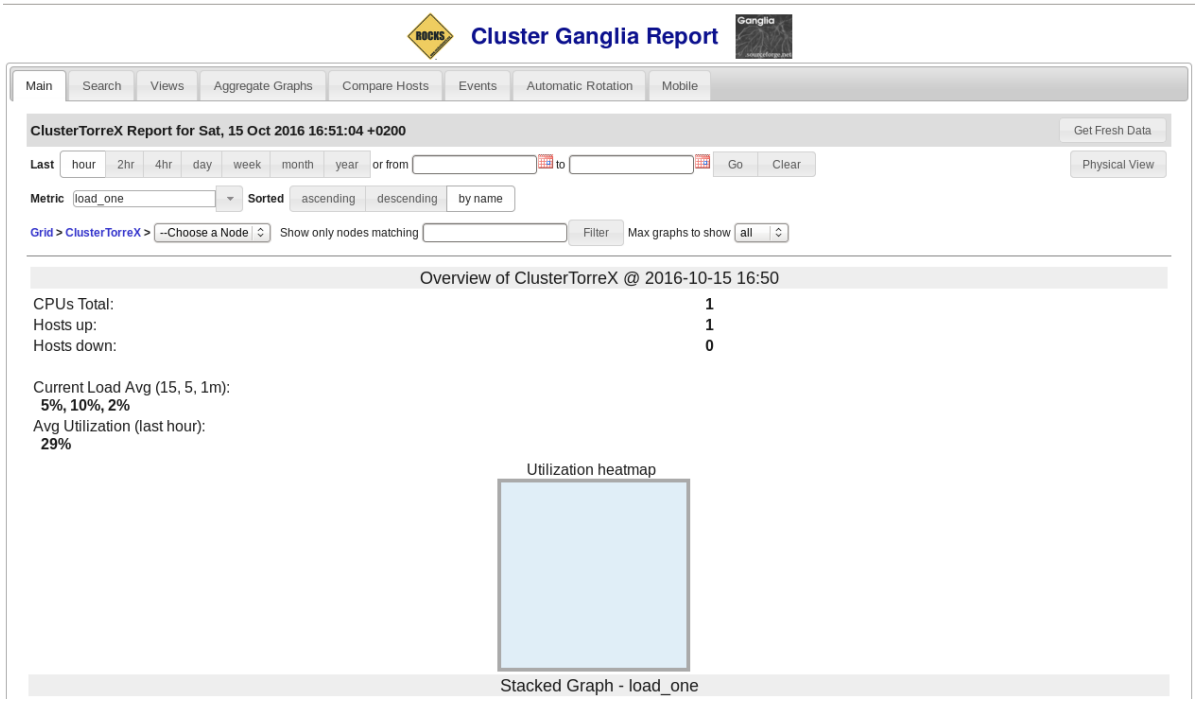


Figura 3.26: Pantalla principal de interface web Ganglia

Ganglia está licenciada a través de BSD como un proyecto open-source original de la Universidad de California, inicialmente llamada Berkeley Millennium Project, fundada en gran parte por el National Partnership for Advanced Computacional Infraestructure (NPACI) y la National Science Fundation.

NPACI está fundada por la National Science Foundation donde algunas investigaciones se centran en desarrollar infraestructuras computacionales basadas en mallas. El soporte actual de Ganglia proviene de PlanetLab, una plataforma abierta para desarrollar, desplegar y facilitar servicios escalables.

El modulo o Roll Ganglia proporciona una interface gráfica de monitorización en directo de los recursos del cluster proporcionados por monitores supervisando cada uno de los hosts. El monitor recoge valores de varias métricas como la carga de la CPU, memoria libre, utilización de disco, entrada/salida red, etc y estas medidas son enviadas a través de la red privada del cluster y son utilizadas en el Frontend para generar las visualizaciones en el panel.

Además, Ganglia proporciona mediante control de pulsos HeartBeat la monitorizacion del estado de servicio de cada nodo, de manera que en caso de pérdida de pulsos por parte de un nodo, este se considerará caído informando a través de la interface mediante distintas representaciones la falta de comunicación.

El monitor se instala seleccionándolo en el proceso de instalación del Frontend, tal como se ha detallado anteriormente, siendo la forma más cómoda de realizarlo, también es posible instalarlo posteriormente. A partir de la instalación el arranque del servicio está automatizado, levantando además un servidor web local que mostrará la información monitorizada en la interface.

El acceso al panel de monitorización se encuentra disponible en la dirección:

http://localhost/ganglia

Este enlace abrirá en el navegador web la página del Roll Granglia, donde se muestra una interface con diferentes opciones para definir la visión de la monitorización completa del sistema.

La interface Ganglia permite monitorizar los recursos hardware de todos los ordenadores del cluster, mostrándolos en diferentes vistas desde donde se pueden obtener los estados de diferentes recursos en función de su utilización en rangos de tiempo o la carga de trabajo total de cada máquina.

Se permite la visión general del estado de los recursos hardware del cluster o diferentes eventos durante el funcionamiento, tanto a nivel individual en nodos como en recursos monitorizados. Estos elementos son mostrados en gráficos que permiten la visualización del uso de cada recurso en un rango de tiempo.

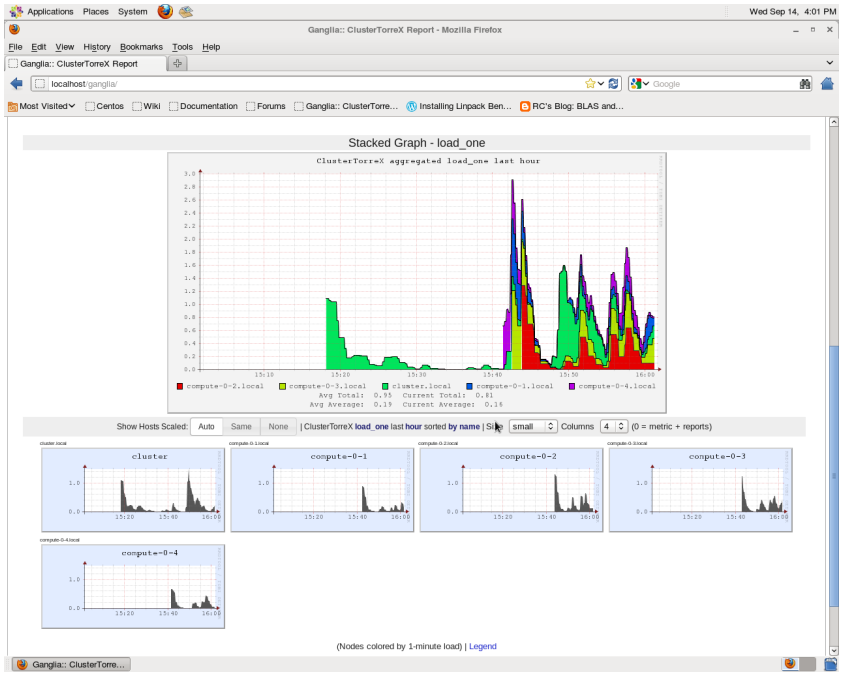


Figura 3.27: Visualización predeterminada del monitor de actividad en nodos del sistema

Como se puede observar en las imágenes de la pantalla principal, se representa cada host con colores diferentes, donde la gráfica muestra los valores de la carga de trabajo total que están recibiendo las máquinas de manera individual en un rango de tiempo establecido.

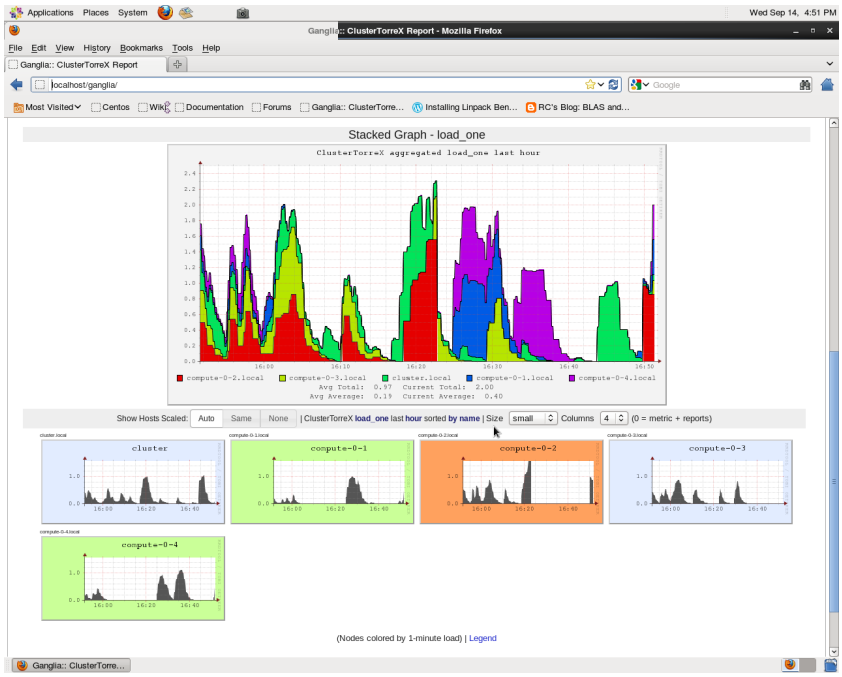


Figura 3.28: Ventana principal de actividad conjunta e individual con coloreado según intensidad de trabajo

También es posible obtener una visión individual o conjunta de determinados recursos, facilitando el control del sistema cluster ayudando a determinar problemas o causas de diferentes fenómenos durante la utilización del sistema. Cada uno de los recursos principales, como CPU, discos, red o memoria, cuentan con diferentes eventos monitorizados por Ganglia, que permitie a su vez la configuración del visor y su representación.

Figura 3.29: Monitorización de la actividad de red en diferentes nodos del cluster

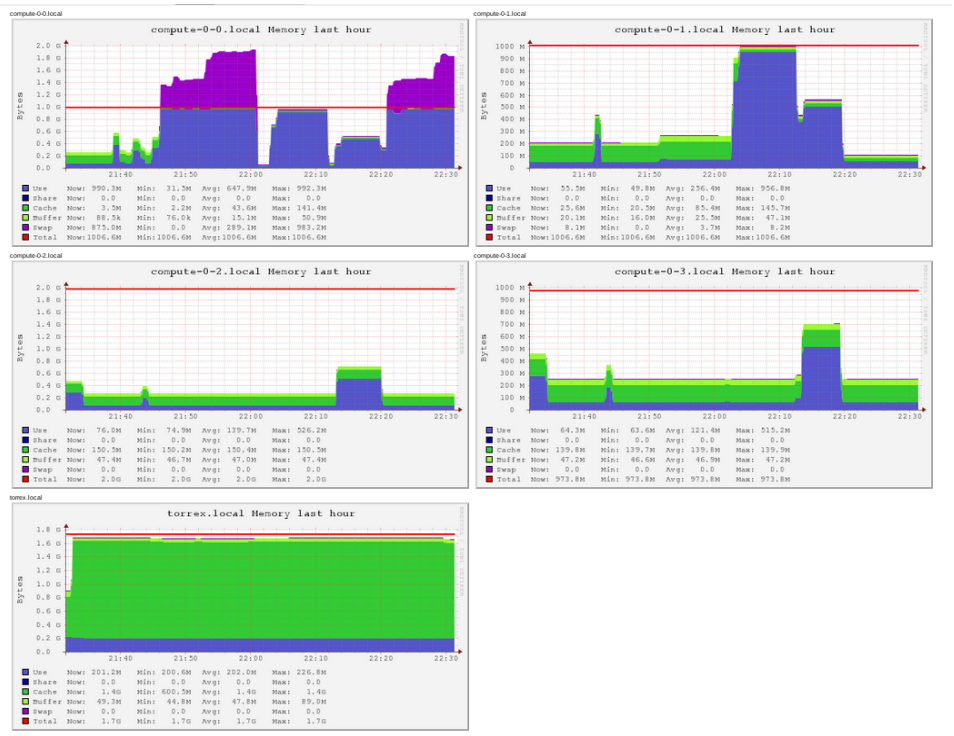


Figura 3.30: Monitorización del estado de memoria RAM en diferentes nodos del cluster

El proceso de monitorización de la actividad de los host también logra detectar sí alguna de las máquinas operativas tiene alguna caída, mediante el método de pulsos Hearthbeat. Esta utilidad es realmente interesante cuando se trabaje en un entorno con múltiples hosts, la cual permitirá detectar inoperatividad por parte de algún nodo con detección prematura de su estado, facilitando la identificación de la máquina exacta.

El servicio es básicamente un demonio activo a nivel de sistema que emite una señal a través de la red cada x tiempo para informar de que el host al que pertenece sigue activo, por ello, en caso de que el host 'muera' o presente algún problema de comunicación con el sistema cluster, Ganglia lo indicará como host caído, notificándolo en la pantalla principal mediante una representación en tonalidad roja sobre la tarjeta del host en cuestión.

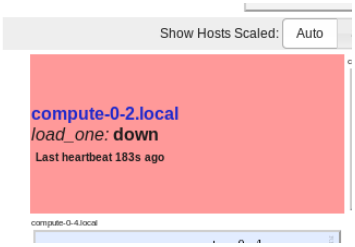


Figura 3.31: Notificación de pérdida de actividad en un nodo del sistema

Ganglia también ofrece una visión general de la capacidad hardware de las máquinas conectadas como hosts al sistema cluster, ofreciendo una visión resumida de los recursos de procesador y memoria de cada nodo así como un sumario de los recursos totales del cluster.

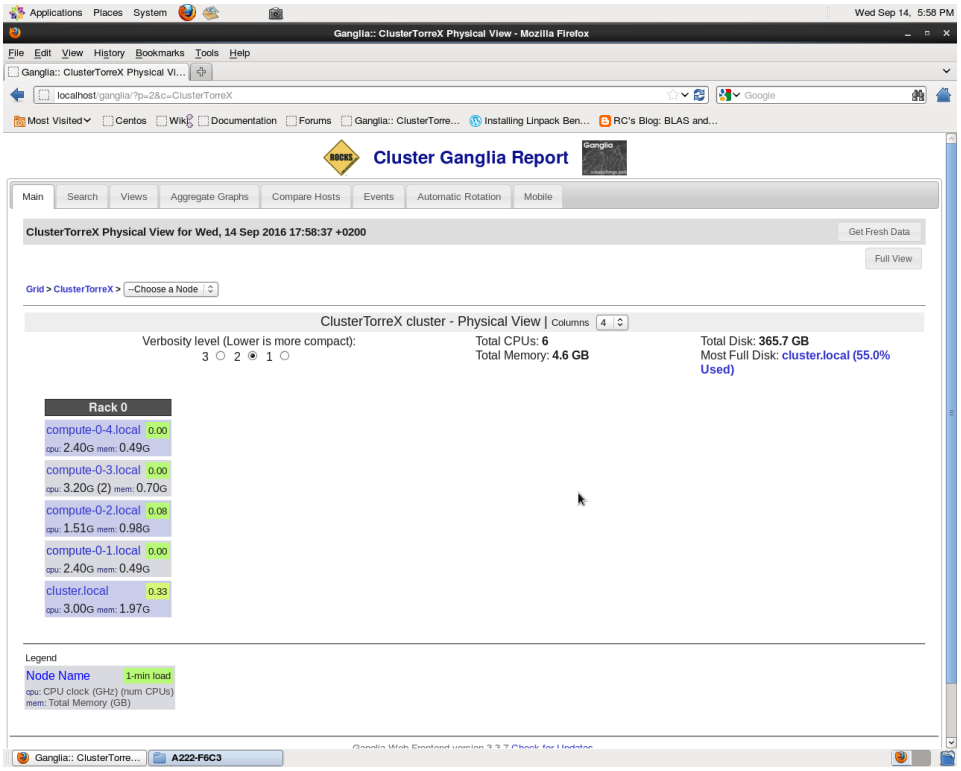


Figura 3.32: Sumario de recursos de CPU y RAM en los nodos del sistema así como a nivel general

También existe la posibilidad de una visión más detallada de cada nodo conectado al cluster, ofreciendo información relacionada con la localización, carga del sistema, utilización del cpu, así como la configuración actual tanto a nivel de hardware como de software de manera individual por máquina.

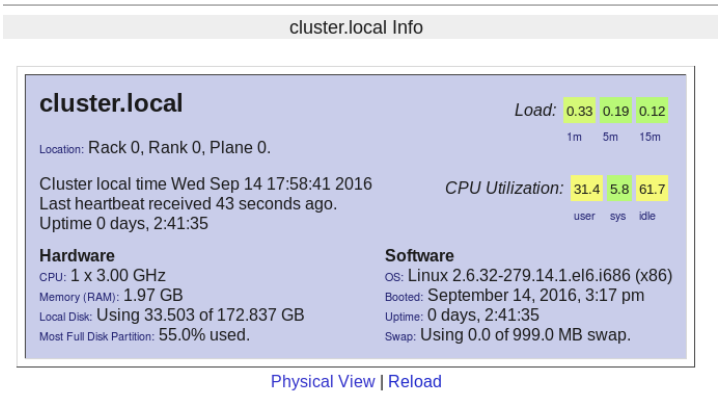


Figura 3.33: Sumario individual de un nodo con sus recursos software/hardware disponibles

Desde el punto de vista de monitorización de métricas sobre los distintos recursos hardware de cada nodo conectado al cluster, Ganglia ofrece la posibilidad de visualizar individualmente distintas métricas relacionadas con recursos principales como cpu, memoria o red, donde en relación a diferentes eventos o valores pueden resultar muy útiles en la gestión y control del sistema cluster y análisis de funcionamiento.

Dentro de estas visualizaciónes se ofrece el poder personalizar la interface, así como la posibilidad de exportar los valores mostrados en las gráficas a ficheros de datos en diferentes formatos como: CSV o JSON, favoreciendo el posterior procesamiento de ellos para estadísticas o análisis.

Figura 3.34: Visualizacion de varias métricas de memoria monitorizadas en un nodo del sistema

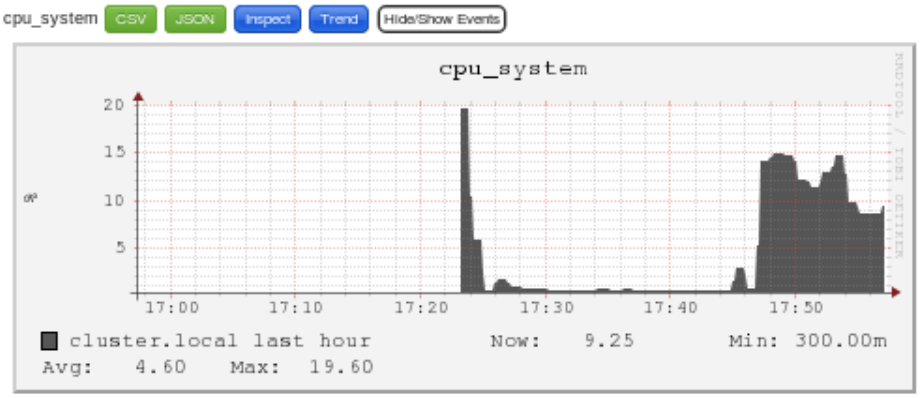


Figura 3.35: Detalle de una metrica y las opciones de exportación de datos en diferentes formatos

3.6.3 Tripwire

Tripwire es un software gratuito Open Source, dentro de la categoría de seguridad para la integración de un sistema de monitorización y alertas en los cambios realizados sobre el sistema de ficheros.

Sus principales funciones están basadas en un sistema de detección de intrusiones basadas en hosts, donde se detectan cambios en los objetos pertenecientes en el sistema de ficheros.

La primera vez que se inicializa, Tripwire escanea el sistema de ficheros y guarda la información de cada fichero escaneado en una base de datos, posteriormente, los mismos ficheros son escaneados y se comparan los resultados con la información almacenada en la base de datos, donde los cambios son reportados al administrador, pudiendo ofrecerse a través de correo electrónico tal como se ha visto. Para detectar los cambios se utilizan técnicas de hashing sin tener que guardar el contenido integro del fichero en la base de datos.

El software Tripwire genera unos informes con todos los detalles relacionados con el escaneo llevado a cabo en el sistema de ficheros, reportando todos los cambios que se han detectado durante la comparación con la información contenida en la base de datos original.

En la distribución Rocks Cluster instalada disponemos de la siguiente versión del software: **Open Source Tripwire(R) 2.4.1**

Se disponen de ficheros de configuración ubicados en las siguientes rutas:

- **/opt/tripwire/etc/tw.pol** - Políticas
- **/opt/tripwire/etc/tw.cfg** - Configuración
- **/opt/tripwire/sbin/tripwire** - Principal logs

A continuación se muestran algunos detalles de los reports generados por esta herramienta

- **Encabezamiento:** se muestra la información de los archivos principales de la monitorización del sistema de ficheros, donde se muestran los hash MD5 de los ficheros principales de Tripwire, para poder controlar que no han sido modificados por una posible intrusión. También se muestran los reports archivados así como la fecha del último report entregado. El encabezamiento tiene el siguiente formato:

```
Tripwire Report for cluster.torrex.org

MD5 Sums of Policy,Config,and Tripwire Executable at Installation:

logger: Tripwire: MD5 :  bd0c1bad73758946ecdbd9b0dcba9c9a /opt/tripwire/etc/tw.pol
logger: Tripwire: MD5 :  1662d2fb29499e25a60553d00ed6615b /opt/tripwire/etc/tw.cfg
logger: Tripwire: MD5 :  123f9adb25475445455ad3ea0d25bfaf /opt/tripwire/sbin/
    tripwire

Archived Reports

August-2016
July-2016

Latest Report as of Mon Aug 22 18:28:02 CEST 2016
```

- **Sumario:** muestra un resumen de la información principal del sistema, hostname, IP y los path donde se encuentran los principales ficheros de configuración.

```
================================================================
Report Summary:
================================================================
Host name:                      cluster.torrex.org
Host IP address:                192.168.1.69
Host ID:                        None
Policy file used:               /opt/tripwire/etc/tw.pol
Configuration file used:        /opt/tripwire/etc/tw.cfg
Database file used:             /opt/tripwire/db/cluster.torrex.org.twd
Command line used:              /opt/tripwire/sbin/tripwire --check --cfgfile /opt/
    tripwire/etc/tw.cfg
```

- **Sistema de ficheros Unix:** Muestra el número de las diferentes modificaciones encontradas en el sistema de ficheros monitorizado en función de las reglas establecidas para la monitorización, a grandes rasgos aquí resume las diferentes categorías donde se engloban los cambios encontrados.

```
---------------------------------------------------------------------------
Section: Unix File System
---------------------------------------------------------------------------
Rule Name                    Severity Level    Added    Removed  Modified
---------                    --------------    -----    -------  --------
Invariant Directories              66              0          0         0
* Tripwire Data Files              100             1          0         0
Critical devices                   100             0          0         0
Tripwire Binaries                  100             0          0         0
User binaries                      66              0          0         0
Kernel Administration Programs     100             0          0         0
Networking Programs                100             0          0         0
System Administration Programs     100             0          0         0
System Information Programs        100             0          0         0
Critical Utility Sym-Links         100             0          0         0
* System boot changes              100             68         5         99
* Security Control                 100             0          0         3
Login Scripts                      100             0          0         0
* Critical configuration files     100             1          1         7
OS executables and libraries       100             0          0         0
Operating System Utilities         100             0          0         0
Shell Binaries                     100             0          0         0
* Libraries                        66              0          0         1
Critical system boot files         100             0          0         0
(/boot)
* Root config files                100             596        0         31

Total objects scanned:  19693
Total violations found:  813
```

- **Object Detail:** En esta sección se detallan todos y cada uno de los objetos que se han encontrado añadidos, modificados o eliminados en el sistema de ficheros, el report incluye información propia de cada uno de los ficheros detectados, englobados dentro de las políticas o categorías mostradas en el punto anterior. Esta sección tiene la mayor parte de la extensión del report, ya que detalla cada uno de los objetos encontrados, a continuación, se mostrarán algunos de los detalles que se muestran de cada cambio encontrado.

  Se introduce cada sección indicando un resumen de los objetos que han sido alterados y el nivel critico de seguridad del tipo de objeto:

```
-----------------------------------------------
Rule Name: System boot changes (/var/run)
Severity Level: 100
-----------------------------------------------
Modified object name:  /var/run/syslogd.pid
Property:            Expected                  Observed
-------------        ----------                ----------
* Inode Number          1573198                   1573205
-----------------------------------------------
Added Objects: 3
-----------------------------------------------
Added object name:  /var/run/gdm/auth-for-gdm-fFEMfE
Added object name:  /var/run/gdm/auth-for-gdm-fFEMfE/database
Added object name:  /var/run/gdm/greeter
-----------------------------------------------
Removed Objects: 4
-----------------------------------------------
Removed object name:  /var/run/gdm/auth-for-gdm-6UXdb5
Removed object name:  /var/run/gdm/auth-for-gdm-6UXdb5/database
Removed object name:  /var/run/gdm/auth-for-root-FxMEVF
Removed object name:  /var/run/gdm/auth-for-root-FxMEVF/database
-----------------------------------------------
Modified Objects: 36
```

  Dentro de la sección de objetos modificados se detalla tanto el nombre del fichero modificado como información sobre la fecha en la que se ha detectado la modificación del archivo.

```
-----------------------------------------------
Modified Objects: 1
-----------------------------------------------
Modified object name:  /etc/rc.d/rocksconfig.d
Property:            Expected                  Observed
-------------        ----------                ----------
* Modify Time          Tue Jul 12 16:43:59 2016    Tue Jul 12 16:48:13 2016
```

  Además, en algunos archivos se reportan algunos detalles propios del objeto, como el tamaño, la fecha de modificación y el hashcode calculado en la fecha anterior y en la actual.

```
Modified object name:  /root/.ICEauthority
Property:            Expected                  Observed
-------------        ----------                ----------
* Size                310                       2170
* Change Time         Tue Jul 12 16:40:56 2016  Mon Aug  1 16:52:45 2016
* CRC32               BXB/+C                    A8TTcw
* MD5                 C24BdOLR0k9FnsLwTQXWlg    Dn/D7HlX7PkmNDHALP6WWp
```

- **Error report:** Al final del report anterior, se dedica una sección a recuento de errores encontrados durante el escaneo del sistema de ficheros, ofreciendo información como el tipo de fichero y la ruta donde se encuentra, el formato es el siguiente:

```
================================================
Error Report:
================================================
------------------------------------------------
Section: Unix File System
------------------------------------------------
1.    File system error.
Filename: /var/lock/subsys/ipmi
No such file or directory
2.    File system error.
Filename: /var/lock/subsys/libvirt-guests
No such file or directory
3.    File system error.
Filename: /var/lock/subsys/mcelogd
No such file or directory
```

Capítulo 4

Cluster Computing: MPI + Benchmarking

En el siguiente capítulo, se detallará el proceso del aprovechamiento computacional del sistema cluster, con la ejecución de programas paralelos sincronizados y desplegados mediante tecnología de paso de mensajes.

Se detallarán los recursos software utilizados, así como su correcta utilización, donde el objetivo principal es poner a prueba la arquitectura implementada para ejecutar sencillos programas para validar su correcto funcionamiento, así como benchmarks más completos para aproximar el rendimiento máximo.

4.1 Objetivos

- Verificar el funcionamiento de la ejecución con modelo de paso de mensajes MPI.
- Ejecución de benchmark sencillo para obtener valores de tiempo de ejecución.
- Ejecución de suite NAS Parallel Benchmark para obtener medidas de rendimiento.
- Conclusiones.

4.2 Benchmarking

Benchmarking es la técnica de evaluar las prestaciones de un computador ejecutando conjuntos de programas que representen una carga de trabajo determinada en las máquinas que se pretende evaluar.

Existen diferentes tipos de benchmarks, que se podrían clasificar en cuatro grupos principales:

- **Aplicaciones reales:** (Compiladores de C, Word, Photoshop...). Pueden presentar problemas de portabilidad relacionados con la dependencia del compilador o del S.O. A veces se utilizan aplicaciones modificadas (scripted applications) para simular interacciones multiusuario complejas (servidores), o quitar operaciones de I/O para medir bien el comportamiento de la CPU.

- **Kernels:** (Bucles de Livermoore, Linpacks). Pequeños trozos de programas reales seleccionados para evaluar características específicas de una máquina o explicar las causas de las diferencias entre máquinas distintas.
- **Simples (Toys):** (Criba de Eratóstenes, Puzzles, Quicksort). Programas pequeños (10-100 líneas), fáciles de escribir, y de resultado conocido.
- **Sintéticos** (Dhrystone, Whetstone). Programas que reproducen los porcentajes de instrucciones y uso de recursos de cargas de trabajo reales.

4.3 Tiempo ejecución: UNIX time

El programa **time** se utilizará para realizar de manera práctica una medida del tiempo que ha tardado un programa en ejecutarse completamente.

La sintaxis del comando es la siguiente:

```
[root@cluster testmpi]# time [options] command [arguments...]
```

Cuando el programa termina, en la salida se detallarán los tiempos estadísticos utilizados en la ejecución del programa, desglosados en los siguientes puntos.

- **User time**
- **System time**
- **Real time**

Los valores de tiempo de ejecución mediante este método resultarán muy útiles cuando se tenga que analizar el rendimiento final de la ejecución de un programa.

4.4 Speedup

En arquitectura de computadores, para poder comparar el incremento de prestaciones en un computador debido a posibles modificaciones realizadas tanto a nivel software como hardware, se puede utilizar el factor denominado ganancia de velocidad o comúnmente denominado **speedup**.

El factor de speedup es definido como la ganancia de velocidad de una misma tarea ejecutada en escenarios distintos a nivel de prestaciones, esta relación obtiene un valor que puede ayudar a determinar el porcentaje de mejora entre las ejecuciones del mismo programa.

La medida de speedup se puede determinar utilizando los tiempos de ejecución de un programa, donde a través de la siguiente formula se puede obtener el valor de ganancia resultante.

$$S_{up} = \frac{T_{sinMejora}}{T_{conMejora}}$$

En la comparativa de rendimiento sobre el sistema cluster, se obtendrá el valor de tiempo de ejecución de cada tarea ejecutada en el sistema con el programa estándar **time** de UNIX tal como se ha detallado. Con los diferentes tiempos de ejecución obtenidos, se podrán calcular

los valores de speedup y aproximar la mejora que existe en la ejecución de un programa entre diferentes condiciones de ejecución.

4.5 MPI: Message Passing Interface

El estándar de paso de mensajes MPI (Message Passing Interface), es una interfaz estándar basada en los consensos del MPI Forum, que incluye más de 40 organizaciones participantes. El objetivo de MPI es establecer un estándar portable, eficiente y flexible para la escritura de programas basados en el paso de mensajes, usualmente ejecutados en entornos de memoria distribuida, donde actualmente brinda soporte también en arquitecturas de memoria compartida e híbridas.

MPI es una especificación para los usuarios y desarrolladores de las librerías de pasos de mensajes. Todo el paralelismo que desarrolla es explicito: es el programador el responsable de identificar correctamente el paralelismo y implementar los algoritmos paralelos utilizando las construcciones MPI.

Está basado en el modelo de programación de paso de mensajes, donde los datos son movidos desde espacios de direcciones de un proceso hacia otro proceso a través de operaciones cooperativas entre ellos.

Las especificaciones de MPI han estado definidas para los lenguajes C y Fortran donde actualmente existen 3 implementaciones distintas.

- MVAPICH - Linux Clusters
- Open MPI - Linux Clusters
- IBM MPI - BG-Q Clusters

El sistema Rocks basado en distribución GNU/Linux, dispone de las implementaciones de MPI: MVAPICH y OpenMPI.

MVAPICH MPI: está desarrollada por la Network-Based Computing Lab de la Universidad estatal de Ohio. Está disponible para la mayoría de sistemas GNU/Linux cluster.

Se puede obtener más información acerca de Open MPI en la página oficial del proyecto:

`http://mvapich.cse.ohio-state.edu/`

OPEN MPI: es una implementación opensource desarrollada y soportada por un consorcio de universidades, investigadores y compañías.

Se puede obtener más información acerca de Open MPI en la página oficial del proyecto:

`https://www.open-mpi.org/`

4.6 Metodología

Principalmente cabe destacar, que el objetivo principal no es incidir en el desarrollo de programas en MPI, más bien, este apartado está orientado a la ejecución de programas basados en esta tecnología obtenidos de fuentes libres para obtener datos relacionados con su ejecución dentro del cluster, para posteriormente, tratar de analizar los resultados en función de variables que pueden afectar al rendimiento.

El método que se empleará, estará dividido de manera aproximada en las siguientes fases:

- Parámetros y compilación
- Ejecución y obtención de datos
- Análisis y conclusiones

4.7 Test MPI: Hello MPI!

Para realizar un test de funcionamiento basado en tecnología MPI, se ejecutará este sencillo programa del estilo de programa Hola Mundo.

Cada proceso ejecutado mostrará en pantalla un mensaje de presentación, identificando que proceso ha sido el que ha llevado a cabo dicha acción.

A continuación se muestra parte del código del programa donde se introducen las anotaciones para el uso de MPI y el desarrollo principal del proceso.

```
#include <stdio.h>
#include <stdlib.h>
#include <mpi.h>

/* Programa 'hola mundo' donde cada procesador requerido se identifica,
basado en ejemplos orginales de Tim Kaiser (http://www.sdsc.edu/~tkaiser),
del San Diego Supercomputer Center, en California */

int main(int argc, char **argv)
{
int myid, numprocs;

MPI_Init(&argc,&argv);
MPI_Comm_size( MPI_COMM_WORLD, &numprocs ); // devuelve el número de procesos
MPI_Comm_rank( MPI_COMM_WORLD, &myid );     // identificacion por el myid asi$

printf("Soy el procesador %d de un total de %d\n",myid,numprocs);

MPI_Finalize();

return 0;
}
```

4.7.1 Compilación y parámetros

Para realizar la compilación del programa se utiliza el siguiente comando invocando al compilador de la implementación elegida de MPI, el cual generará el programa ejecutable.

Bastaría con ubicar la shell en el directorio donde disponemos del código fuente y ejecutar el siguiente comando:

```
[zxcv90@cluster testmpi]$ mpicc testMPI.c -o testMPI
```

Esto generará el binario que se podrá ejecutar utilizando la directriz **mpirun**, definiendo los parámetros necesarios explicados en el siguiente apartado.

4.7.2 Ejecución

La ejecución de programas en MPI, utiliza el comando **mpirun** utilizada por las implementaciones instaladas en el sistema.

El comando necesita dos parámetros muy importantes que determinan la ejecución del binario en el entorno:

- **-np** Define el número de procesos que se desplegarán durante la ejecución del programa.
- **-machinefile (archivo de hosts)** Este parámetro lee un fichero donde se definen las máquinas, nodos o host donde se ejecutará el programa MPI con el número de procesos asignados en el parámetro anterior.

El parámetro **-np** define los threads de ejecución del programa, estos threads pueden desplegarse en el sistema cluster sobre varias máquinas definiendo previamente el fichero que se pasará como parámetro en **-machinefile**, que contrendrá los nombres asignados a cada nodo.

El contenido del fichero leido en el parámetro **-machinefile** podría ser el siguiente:

```
compute-0-0
compute-0-1
compute-0-2
compute-0-3
```

Este fichero se puede guardar con el nombre **machines** y será modificado siempre que se quieran cambiar las máquinas que ejecutarán el programa, tambien se puede disponer de varios archivos para posteriores programaciones de scripts y ejecución mediante SGE dependiendo de máquinas implicadas.

La ejecución del programa anterior en los 4 nodos del cluster disponibles se realizaría con el siguiente comando:

```
[zxcv90@cluster testmpi]$ mpirun -np 4 -machinefile machines testMPI
```

Donde la salida del programa sería como lo siguiente:

```
[root@cluster testmpi]# mpirun -np 4 testMPI
Soy el procesador 1 de un total de 4
Soy el procesador 2 de un total de 4
Soy el procesador 0 de un total de 4
Soy el procesador 3 de un total de 4
```

Esta salida sin errores indica que la ejecución paralela del programa anterior ha sido satisfactoria.

4.8 Simple Toy Benchmark: Prime Numbers MPI

Un número primo es un número natural que tiene exactamente dos divisores, el mismo y 1, existen infinitos números primos, el número primo más pequeño es 2.

El siguiente apartado está basado en un programa escrito en C que desarrolla el algoritmo de obtención de los números primos de un número, pudiendo ejecutarse en versión paralela MPI, donde se repartirá la carga de trabajo en las distintas máquinas implicadas, que realizarán la búsqueda y las comprobaciones de los distintos números.

4.8.1 Objetivos

El objetivo principal de este apartado no es ofrecer una ejecución eficiente o mejora del rendimiento del algoritmo mediante programación.

El objetivo principal es obtener el tiempo total de ejecución para diferentes tallas de problemas sobre varios nodos, para posteriormente comparar dichos resultados y obtener alguna valoración.

Para proceder, se obtendrá el tiempo de ejecución con el comando: **time**, que ofrecerá al finalizar la ejecución del programa un resumen del tiempo necesario para la ejecución total del programa.

Se realizará la ejecución para diferentes tallas de problema, incrementando exponencialmente el número máximo para el que se tendrán que buscar los números primos que contiene.

Los diferentes valores, siguiendo un crecimiento exponencial son los siguientes:

Problema (N)	Valor
2^{18}	262144
2^{19}	524288
2^{20}	1048576
2^{21}	2097152
2^{22}	4194304

Tabla 4.1: Tamaños de problema con incremento exponencial

Estos valores son modificables en el código fuente del programa, concretamente en la variable **n_hi**, que determina el número máximo para el que se realiza la búsqueda.

```
n_hi = 1048576 //Núm max
```

Después de cada modificación en el código es necesario recompilarlo, además de que dependiendo de las máquinas que intervendrán en la resolución del cálculo, se debería de modificar tanto el parámetro -np como editar el fichero **machines** para incluir los hosts donde se ejecutará el programa.

Para concluir una comparativa del rendimiento se ejecutará en versión paralela en diferentes máquinas del cluster, para poder percibir sí existe mejora en la división de la carga del cálculo sobre las diferentes máquinas que componen el la arquitectura.

4.8.2 Algoritmo paralelizado

El algoritmo es sencillo. Para cada entero i, se verifica sí alguna J divide de manera uniforme el entero, la cantidad de trabajo está definida por N que es el número máximo para el que se tienen que obtener los primos menores que el.

El objetivo de la ejecución paralela, es que la cantidad de trabajo N, se divida entre los procesos que se ejecutarán en el programa, de manera que cada uno de ellos compruebe los números primos incluidos dentro de un rango acotado. Este rango estará compuesto por números obtenidos del número máximo N recorrido en incrementos definidos por el número de proceso, determinando pares o impares en función del identificador del thread.

De esta manera, para encontrar los números primos de N, se podrían encontrar diferentes despliegues determinados por el número de procesos desplegados por el programa.

En el reparto de trabajo de cálculo entre los procesos, afectan directamente las variables **id (identificador del proceso)**, usualmente incremento nominal desde 0, y el número de procesos **p**, que determina la próxima iteración para la comprobación de primos.

```
for ( i = 2 + id; i <= n; i = i + p ){
```

Ejemplo carga de trabajo 2 procesos:

Números comprobados por 2 procesos. En la primera iteración se define: **proceso (0)**, 2+0(id)=2, analizará los números pares, **proceso(1)**, 2+1(id)=3, analizará números impares. Próxima iteración definida por el número de procesos, en este caso 2.

Proceso 0 (id 0)	Proceso 1 (id 1)
2	3
4	5
6	7
8	9
10	11
12	13
14	15
16	17
18	19
20	

Tabla 4.2: Números que analizaran 2 procesos.

Ejemplo carga de trabajo 4 procesos:

Números comprobados por 4 procesos. En la primera iteración se define: **proceso(0)** 2+0(id)=2, analizará números pares, **proceso(1)** 2+1(id)=3, analizará números impares, **proceso (2)**, 2+2(id)=4, analizará números pares, **proceso(3)** 2+3(id)=5, analizará números impares. Incremento definido por el número de procesos, 4.

Con el rango definido para cada proceso, el algoritmo realiza una iteración interna para cada número para comprobar los números primos que incluye cada uno. Esto se consigue descartando aquellos números en los que el resto de la división entre los dos elementos comprobados sea 0.

Proceso 0 (id 0)	Proceso 1 (id 1)	Proceso 3 (id 2)	Proceso 4 (id 4)
2	3	4	5
6	7	8	9
10	11	12	13
14	15	16	17
18	19	20	

Tabla 4.3: Números que analizaran 4 procesos.

4.8.3 Ejecución paralela: Prime MPI

En la ejecución paralela entre las máquinas que conforman el cluster, la cantidad de trabajo total que tiene que desarrollar el programa, es dividida entre el número de procesos activos en cada nodo de cálculo para obtener su resolución tal como se ha explicado. El número de procesos definirá las iteraciones que realiza la obtención de primos, por tanto, definirá la carga de trabajo que se asignará a cada proceso en cada máquina.

A continuación se muestra parte del código fuente del algoritmo paralelo donde se realiza la obtención de números primos.

Dentro de este fragmento de código se pueden identificar las siguientes variables:

Variables:

- **id**= Identificador del proceso
- **p**= Cantidad de procesos ejecutados
- **n**= Rango de búsqueda de números primos

A continuación, el fragmento de código donde aparecen las anotaciones del estándar MPI, que definen el procesamiento entre las diferentes máquinas utilizando la tecnología de paso de mensajes.

```
while(n <= n_hi){

  if(id == 0){
    wtime = MPI_Wtime();
  }

  ierr = MPI_Bcast(&n,1,MPI_INT,0,MPI_COMM_WORLD);

  primes_part = prime_number(n,id,p);

  ierr = MPI_Reduce(&primes_part,&primes,1,MPI_INT,MPI_SUM,0, MPI_COMM_WORLD);

  if(id == 0){
    wtime = MPI_Wtime() - wtime;
    printf ( "  %8d  %8d  %14f\n", n, primes, wtime );
  }

  n = n * n_factor;
}

/*Terminate MPI.*/
ierr = MPI_Finalize();
```

En el siguiente fragmento de código se muestra la función que realiza el algoritmo para definir sí un número es primo o no y contabilizarlo globalmente para el resultado final del proceso. La variable **p**, con el número de procesos activos, define las iteraciones a lo largo de los números analizados, y por ende, la carga de trabajo de cada thread.

```
for (i = 2 + id; i <= n; i = i + p){

  prime = 1;

  for (j = 2; j < i; j++){
    if ((i %j) == 0){

      prime = 0;
      break;
    }
  }

  total = total + prime;
}
```

Las máquinas utilizadas en la ejecución del algoritmo son las siguientes:

Node	Mhz
Compute-0-0	2399,94
Compute-0-1	2399,94
Compute-0-2	2800
Compute-0-3	2800

Tabla 4.4: Frecuencia de los procesadores utilizados

4.8.4 Performance

Después de realizar la ejecución del programa en diferentes tamaños de problema y variando el número de máquinas que intervienen en la resolución del algoritmo de forma paralela, los resultados obtenidos se muestran en la siguiente tabla.

Talla (N)	1 Nodo	2 Nodos	3 Nodos	4 Nodos
262144	0m55.227s	0m55.191s	0m46.124s	0m28.466s
524288	3m24.892s	3m22.958s	2m48.662s	1m42.230s
1048576	12m44.401s	12m45.566s	10m35.578s	6m23.755s
2097152	48m33.769s	48m22.689s	40m19.352s	24m9.671s

Tabla 4.5: Tiempo de ejecución del programa en N nodos

A continuación, en diferentes gráficas se mostrará la representación lineal del tiempo total de ejecución en diferentes tamaños de problema sobre diferentes configuraciones en número de nodos implicados en la ejecución.

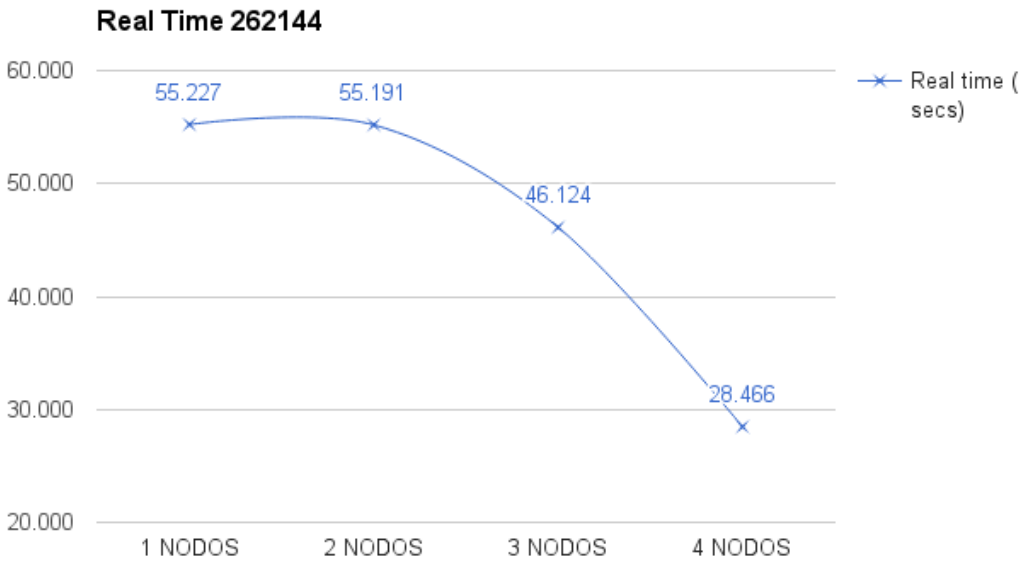


Figura 4.1: Tiempos de ejecución para carga de trabajo 2 [18]

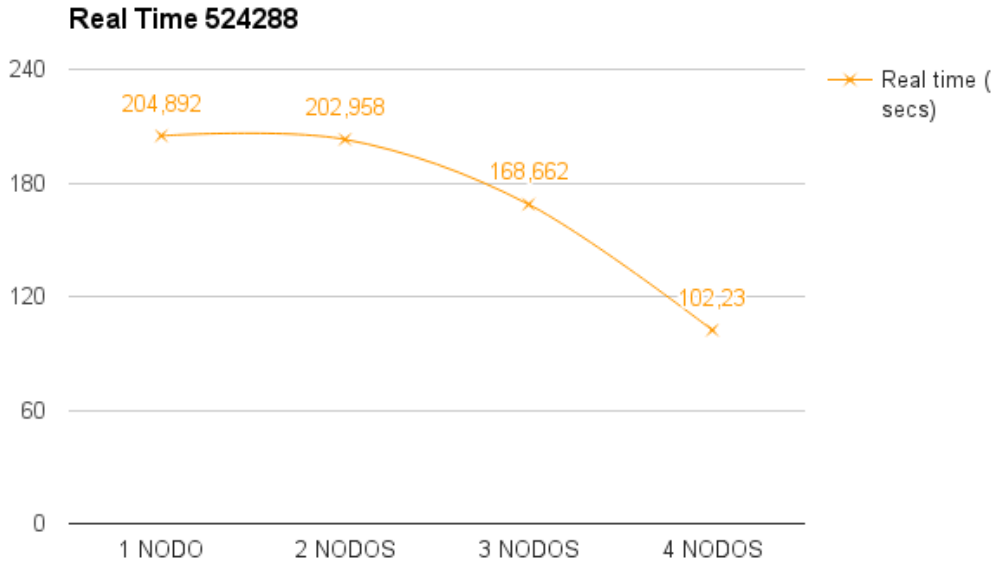


Figura 4.2: Tiempos de ejecución para carga de trabajo 2 [19]

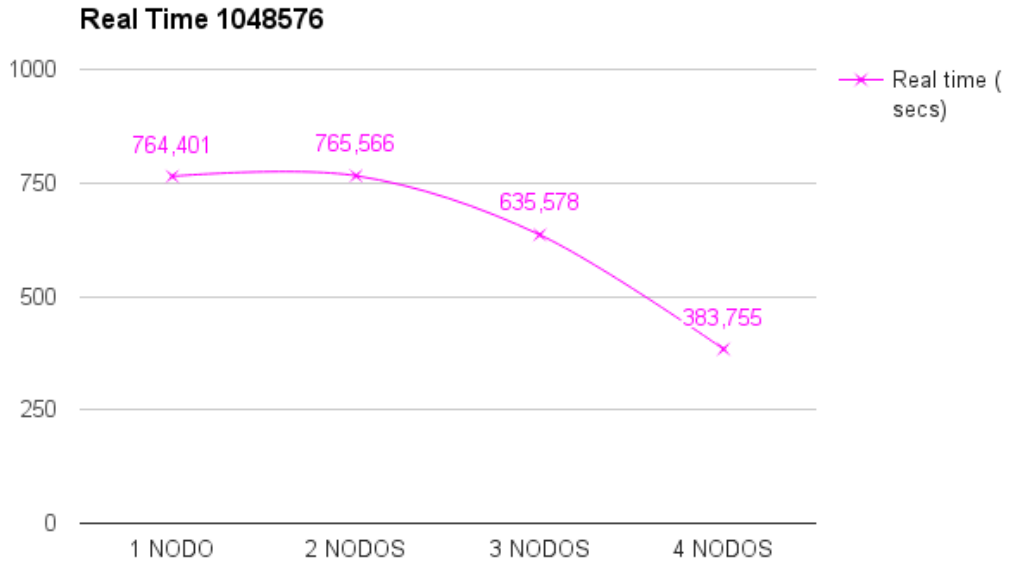


Figura 4.3: Tiempos de ejecución para carga de trabajo 2 [20]

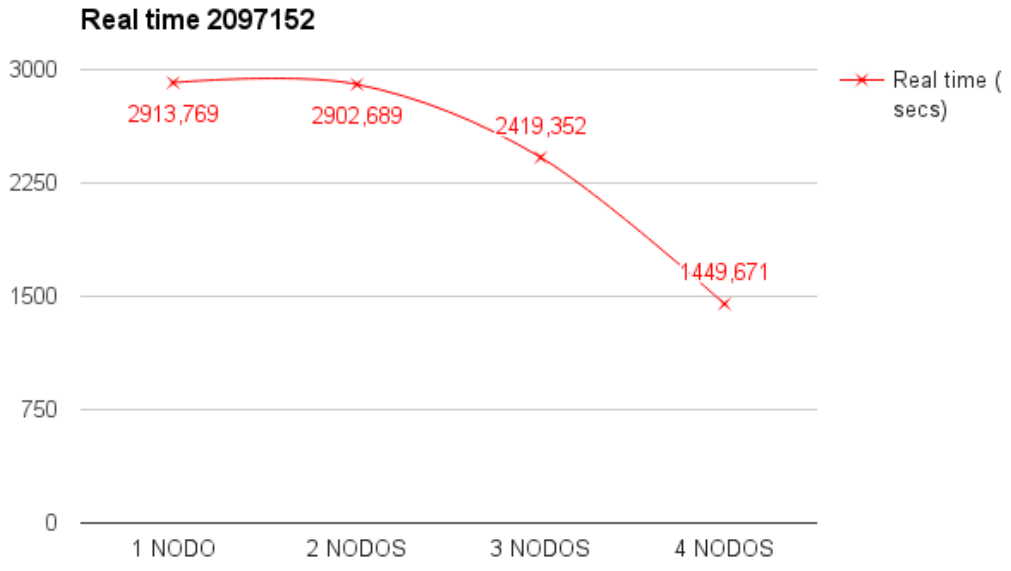


Figura 4.4: Tiempos de ejecución para carga de trabajo 2 [21]

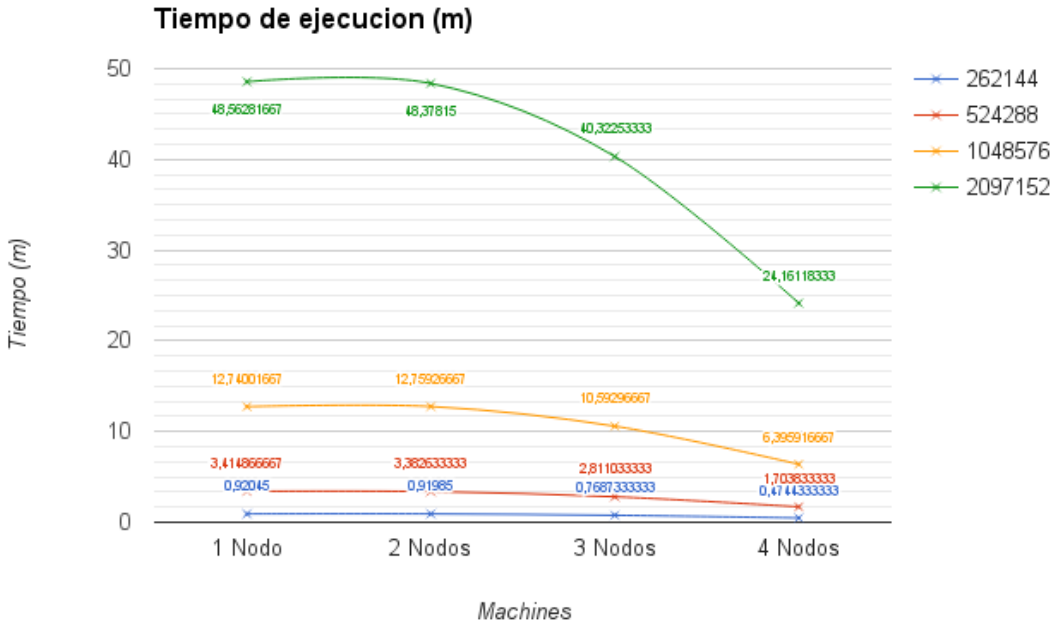


Figura 4.5: Tiempos de ejecución de las diferentes ejecuciones

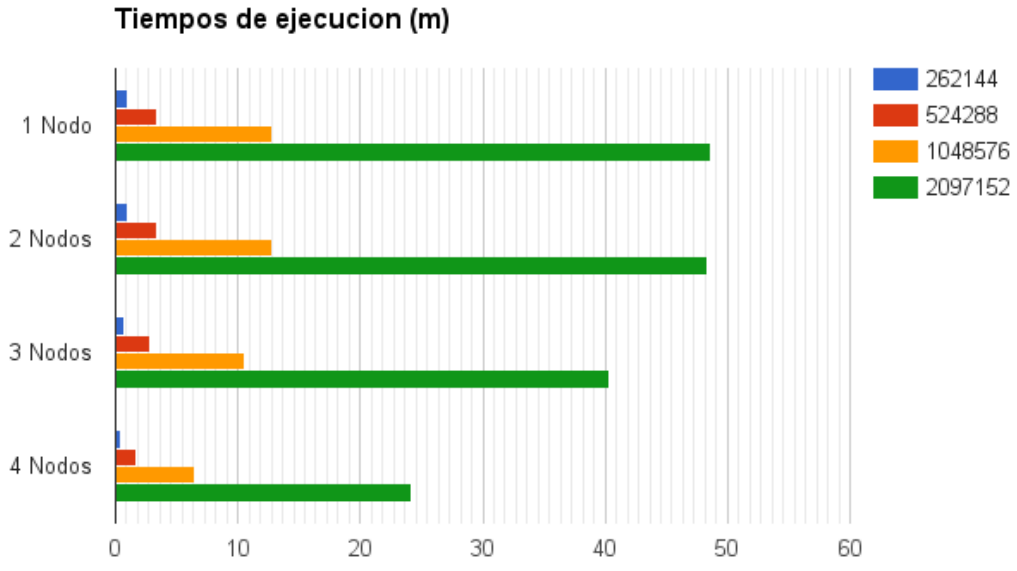


Figura 4.6: Tiempos de ejecución de las diferentes ejecuciones en gráfica de barras

4.8.5 Speedup

En el siguiente apartado, se obtendrán los valores de speedup correspondientes a los tiempos mesurados para poder obtener el ratio de mejora en tiempo de ejecución entre las diferentes configuraciones de número de nodos y talla de problema. Los valores se verán representados tanto en tabla como en gráfica lineal de ratio de mejora.

Los valores de speedup calculados están detallados en la siguiente tabla representados como la relación entre 1 y N máquinas.

Talla (N)	1/2 Nodos	1/3 Nodos	1/4 Nodos
262144	1,00	1,20	1,94
524288	1,01	1,21	2,00
1048576	1,00	1,20	1,99
2097152	1,00	1,20	2,01

Tabla 4.6: Speedup del programa Prime Numbers en 4 nodos

Como puede observarse, el ratio de mejora se mantiene constante en la relación entre el número de máquinas implicadas y el incremento en la carga de trabajo del problema planteado. Se alcanza un ratio de mejora máximo de **2.00** puntos de speedup, cuando se compara la ejecución del programa en todas las máquinas disponibles del sistema cluster frente a la ejecución en una sola máquina. Teóricamente se podría esperar una mejora de **4x** más velocidad, pero en la práctica esta relación no se alcanza en este caso.

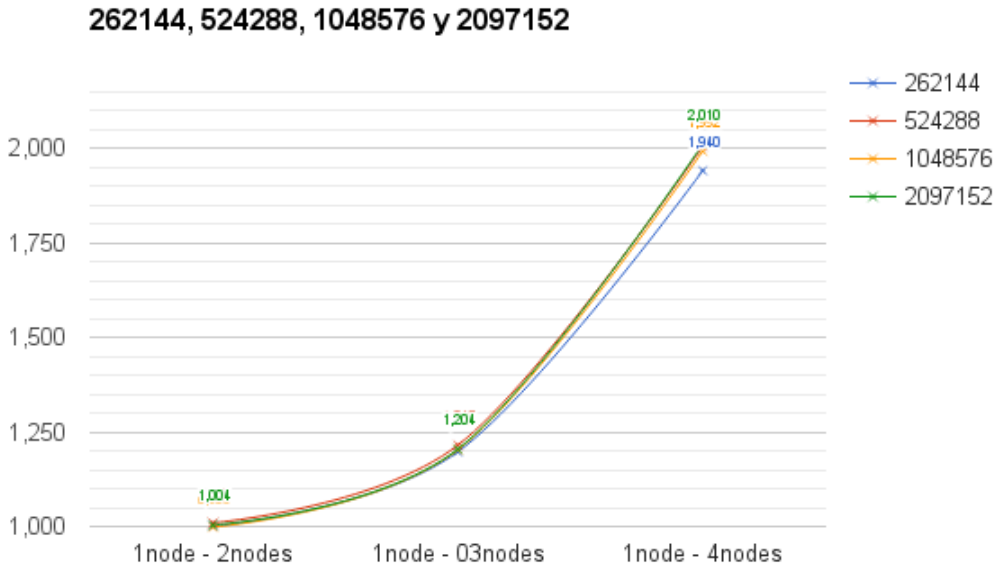


Figura 4.7: Representación gráfica del incremento de speedup

4.9 Benchmarking: NAS Parallel Benchmarks

NAS Parallel Benchmarks de ahora en adelante (NPB), es un set de programas para ayudar en el proceso de evaluación de entornos de supercomputación paralelos. Los benchmarks son derivados de aplicaciones orientadas a la computación de dinámicas de fluidos (CFD) y consisten en 5 kernels y tres pseudo-aplicaciones en su versión básica inicial. Esto es debido a que los desarrolladores consideraron que los kernels por sí solos no son suficientes para evaluar totalmente el rendimiento de la máquina, por lo que se incluyen una serie de pseudo-aplicaciones para simular un cálculo real de dinámicas de fluidos.

La suite de benchmarking expuesta, incluye en los benchmarks cálculos para adaptación de mallas desestructuradas, entrada/salida paralela, aplicaciones multi-zona y mallas computacionales, donde el tamaño de los problemas en NPB están definidos e indicados como clases diferentes, de esta manera resulta más sencillo analizar de forma gradual el comportamiento de una aplicación frente a diferentes cargas de trabajo.

Las implementaciones están desarrolladas tanto en Fortran90 como en C, donde se ofrece soporte a modelos de programación como MPI o OpenMP.

Los kernels y pseudo-aplicaciones utilizados aquí, involucran substancialmente procesos de computación más costosos y acordes a la tecnología de computadores actuales que benchmarks anteriores como el conocido Linpack o Livermore Loops, por lo que resultan más apropiados para la evaluación de máquinas paralelas actualizadas o de reciente generación.

En el caso expuesto, la evaluación de prestaciones será sobre el modelo de programación MPI, en diferentes kernels básicos más acordes a la infraestructura disponible.

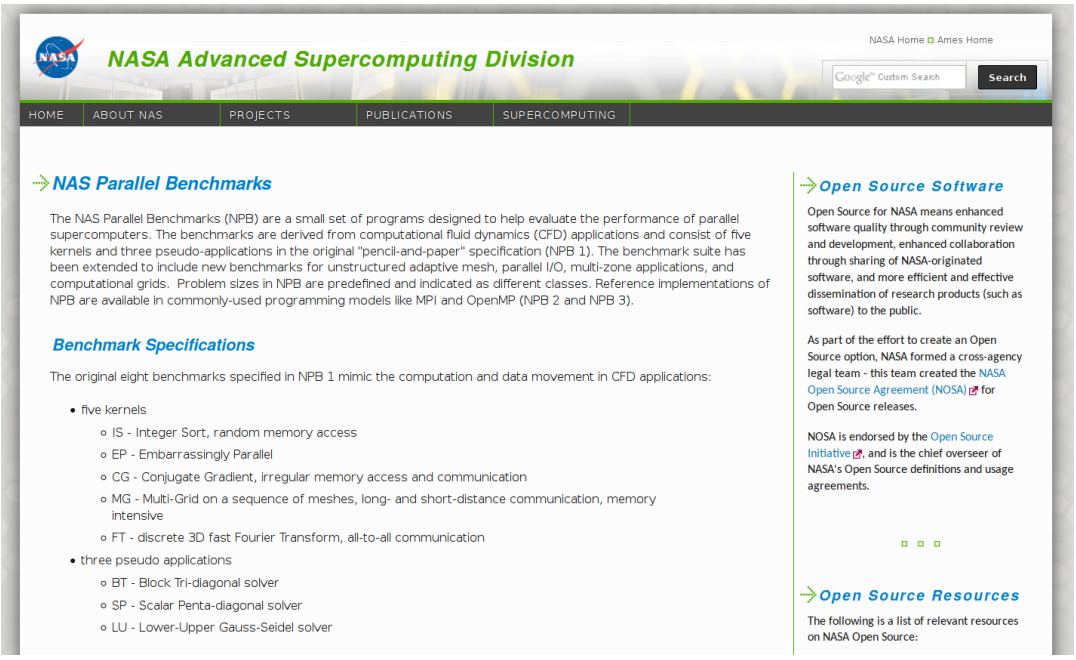


Figura 4.8: Portal web del proyecto NAS Parallel Benchmarks

4.9.1 Objetivos

El objetivo será la introducción a la suite de benchmarks NAS, dándola a conocer como un recurso útil para la medida de rendimiento en entornos de supercomputación, debido a su característica de basarse en problemas muy aproximados a la realidad.

El punto principal se basará en la obtención del tiempo total de ejecución de los Kernels originales de la suite NAS Parallel Benchmarks, ejecutando los programas en diferentes nodos y cargas de trabajo para posteriormente comparar dichos resultados.

4.9.2 Especificaciones de los benchmarks

Los 8 benchmarks especificados en la primera versión de NPB están orientados a resolver mediante computación paralela el movimiento de datos en aplicaciones de cálculo de dinámicas de fluidos, CFD. A estos benchmarks iniciales se añadieron posteriormente diferentes programas nuevos para aproximar más a la realidad la medida de rendimiento de un sistema paralelo bajo diferentes problemas o condiciones. A continuación se realizará una explicación breve de los distintos objetivos de cada aplicación, así como un resumen de su propósito.

Kernels:

- **IS - Integer Sort**. Ordena N claves en paralelo. Las claves son generadas por un algoritmo de generación de claves y distribuidas en memoria. La distribución inicial de las claves puede tener un gran impacto en el rendimiento. Este tipo de clasificación es importante en los códigos de métodos de partículas, donde pone a prueba tanto el nivel de carga computacional y el rendimiento de la comunicación.
- **EP - Embarassing Parallel**. Kernel de paralelismo embarazoso. Proporciona una estimación de los limites superiores alcanzables en rendimiento de coma flotante. Genera pares de desviaciones aleatorias gaussianas de acuerdo con un esquema específico. Es un problema tipico en algunas aplicaciones de simulación como Montecarlo. El único requisito de comunicación es la combinación de 10 sumas al final del programa, donde las secciones separadas de los números aleatorios pueden ser independientes y calculadas en procesadores separados.
- **CG - Conjugate Gradient**. Este benchmark, utiliza el método de potencia inversa para encontrar una estimación del valor propio más grande de una matriz dispersa definida simétricamente con un patrón aleatorio de valores diferentes a 0. Está orientado al acceso a memoria irregular y comunicación.
- **MG - Multigrid**. El kernel ejecuta cuatro iteraciones del algoritmo multigrid V-cycle para obtener una solución aproximada al problema discreto de Poisson. Realiza pruebas tanto en envío de datos en corta y larga distancia.
- **FT - discretización 3D de la Transformada de Fourier**. Una solución 3D parcial de ecuaciones diferenciales utilizando la Transformada de Fourier. Es un test riguroso en el rendimiento de comunicación de larga distancia para interconexión de clusters.

Pseudo - Aplicaciones

- **BT** - Cálculo de Bloques Tri-diagonales.
- **SP** - Cálculo de Escalares Penta-Diagonales.
- **LU** - Cálculo Lower - Upper Gauess - Seidel.

Versión multizona de las pseudo-aplicaciones NPB

Estos benchmarks basados en las pseudo-aplicaciones listadas anteriormente, están diseñadas para explotar múltiples niveles de paralelismo en aplicaciones y probar la eficiencia de una paralelización híbrida o multinivel entre diferentes herramientas y paradigmas.

Estos benchmarks se caracterizan por basar su carga de trabajo en diferentes zonas, que luego son divididas en diferentes subzonas más pequeñas para un planteamiento paralelo del problema inicial.

- **BT-MZ** - Cálculo de Bloques Tri-diagonales en zonas de tamaño no uniforme dentro de una talla de problema, aumento del número de zonas a medida que crece la talla de problema.
- **SP-MZ** - Cálculo de Escalares Pentadiagonales en zonas de tamaño uniformes dentro de una talla de problema, aumento del número de zonas a medida que crece la talla de problema.
- **LU-MZ** - Cálculo Lower - Upper Gauess - Seidel en zonas de tamaño no uniforme dentro de una talla de problema, número fijo de zonas para todas las tallas del problema.

Benchmarks de computación desestructurada, E/S paralela o movimiento de datos.

- **UA - Malla desestructurada adaptativa**. Este benchmark involucra la solución de un problema de transferencia de calor en un dominio cúbico discretizado en una malla desestructurada.
- **BT-IO - Basado en el benchmark BT inicial**. Este benchmark está orientado a medir las capacidades de salida de un computador de alto rendimiento. Donde cada procesador es responsable de escribir la fracción de datos asignada en un fichero de salida.
- **DC - Data Cube**. Benchmark orientado a la prueba de movimiento de sets de datos distribuidos. Orientado a testear la jerarquía de memoria en todo el sistema.
- **DT - Data Traffic**. Benchmark orientado a la medida del rendimiento en el trafico de datos a través de la red.

4.9.3 Clases

Las clases en NPB definen los tamaños de problema que afrontarán los distintos programas durante su resolución, clasificándose en 4 categorías principales, donde en dos de ellas se encuentran incrementos exponenciales del tamaño de problema en 3 grados diferentes. Son las siguientes:

- **Class S** - tamaño pequeño para test rápidos
- **Class W** - tamaño workstation, (orientado a una estación de trabajo de los años 90, actualmente tamaño intermedio)
- C**lass A, B, C** - test de problemas estándar; incremento de 4x aprox. entre una clase y la siguiente
- **Class D, E, F** - test de problemas estándar; incremento de 16x aprox. entre una clase y la siguiente

El sumario de los tamaños de problema y los parámetros definidos son los siguientes:

- **Kernel IS**

Parámetro	S	W	A	B	C	D	E
núm de claves	2^{16}	2^{20}	2^{23}	2^{25}	2^{27}	2^{31}	
valor máximo clave	2^{11}	2^{16}	2^{19}	2^{21}	2^{23}	2^{27}	

Tabla 4.7: Tamaños de problema para Kernel IS

- **Kernel EP**

Parámetro	S	W	A	B	C	D	E
núm de pares aleatorios	2^{24}	2^{25}	2^{28}	2^{30}	2^{32}	2^{36}	2^{40}

Tabla 4.8: Tamaños de problema para Kernel EP

- **Kernel CG**

Parámetro	S	W	A	B	C	D	E
núm de filas	1400	7000	14000	75000	150000	1500000	9000000
núm de nonzeros	7	8	11	13	15	21	26
núm de iteraciones	15	15	15	75	75	100	100
eigenvalue shift	10	12	20	60	110	500	1500

Tabla 4.9: Tamaños de problema para Kernel CG

- **Kernel MG**

Parámetro	S	W	A	B
Tamaño malla	32x32x32	128x128x128	256x256x256	256x256x256
núm de iteraciones	4	4	4	20

Tabla 4.10: Tamaños de problema para Kernel MG

Parámetro	C	D	E
Tamaño malla	512x512x512	1024x1024x1024	2048x2048x2048
núm de iteraciones	20	50	50

Tabla 4.11: Tamaños de problema para Kernel MG

- **Kernel FT**

Parámetro	S	W	A	B
Tamaño malla	64x64x64	128x128x32	256x256x128	512x256x256
núm de iteraciones	6	6	6	20

Tabla 4.12: Tamaños de problema para Kernel FT

Parámetro	C	D	E
Tamaño malla	512x512x512	2048x1024x1024	4096x2048x2048
núm de iteraciones	20	25	25

Tabla 4.13: Tamaños de problema para Kernel FT

4.9.4 Descarga del Bennchmark

La suite de benchmarks NPV se puede descargar desde el siguiente link:

`https://www.nas.nasa.gov/assets/npb/NPB3.3.1.tar.gz`

Para la ejecución es necesario descargarlo y ubicarlo en una carpeta que se considere y donde se dispongan de permisos de administrador.

Actualmente está disponible la versión NPB 3.3.1.

4.9.5 Configuración inicial fichero make.def

Una vez descargado el fichero y extraido el código fuente de la suite, se debería de configurar y compilar para generar los binarios de los programas benchmark que posteriormente se ejecutarán en el sistema.

Previamente a la compilación, se deberán de configurar algunos parámetros necesarios en las variables que se utilizarán en la compilación. Esta configuración se aplica sobre el fichero **make.def**, ubicado en el directorio **NPB3.3-MPI/config**, en caso de no existir, se puede copiar la plantilla make.def.template y editar el contenido para las características especificas del sistema deseado.

A continuación se muestran las líneas del fichero make.def que se deberían de configurar para una compilación nueva de la suite:

Compilador de Fortran predefinido:

```
#---------------------------------------------------------------------
# This is the fortran compiler used for MPI programs
#---------------------------------------------------------------------
MPIF77= mpif77
# This links MPI fortran programs; usually the same as ${MPIF77}
FLINK= $(MPIF77)
```

Compilador de C predefinido:

```
#---------------------------------------------------------------------------
# This is the C compiler used for MPI programs
#---------------------------------------------------------------------------
MPICC= mpicc
# This links MPI C programs; usually the same as ${MPICC}
CLINK= $(MPICC)
```

```
#---------------------------------------------------------------------------
# MPI dummy library:
#
# Uncomment if you want to use the MPI dummy library supplied by NAS instead
# of the true message-passing library. The include file redefines several of
# the above macros. It also invokes make in subdirectory MPI_dummy. Make
# sure that no spaces or tabs precede include.
#---------------------------------------------------------------------------
# include ../config/make.dummy
```

Utilidades de C:

```
#---------------------------------------------------------------------------
# Utilities C:
#
# This is the C compiler used to compile C utilities.  Flags required by
# this compiler go here also; typically there are few flags required; hence
# there are no separate macros provided for such flags.
#---------------------------------------------------------------------------
CC= cc -g
```

Destino de los ejecutables compilados:

```
#---------------------------------------------------------------------------
# Destination of executables, relative to subdirs of the main directory. .
#---------------------------------------------------------------------------
BINDIR= ../bin
```

Variable de control en la generación de números aleatorios:

```
#---------------------------------------------------------------------------
# The variable RAND controls which random number generator
# is used. It is described in detail in Doc/README.install.
# Use "randi8" unless there is a reason to use another one.
# Other allowed values are "randi8_safe", "randdp" and "randdpvec"
#---------------------------------------------------------------------------
RAND= randi8
```

Existen más opciones de configuración en el archivo que ofrecen la flexibilidad de adecuar la generación de los binarios dependiendo de las necesidades que se requiera a nivel de optimización. De manera básica, con la configuración anterior es posible ya realizar la compilación correcta de los binarios ejecutables.

4.9.6 Compilación

Para realizar la compilación de los benchmarks de la suite, se utilizará el comando **make**. La sintaxis para la compilación de cualquier benchmark es la siguiente:

```
make <benchmark-name> NPROCS=<number> CLASS=<class> [SUBTYPE=<type>] [VERSION=VEC]
```

Donde se deberán de definir los siguientes argumentos:

- **benchmark-name** - corresponderá al benchmark que se desee compilar: bt, cg, dt, ep, ft, is, lu, mg, o sp
- **number** - es el número de procesos que ejecutarán con el benchmark
- **class** - es la clase, descrita anteriormente, definirá el tamaño del problema que afrontará el programa en ejecución: S, W, A, B, C, D, E

Las clases C, D y E no están disponibles para el benchmark DT.

La opción VERSION=VEC es utilizada para seleccionar las versiones vectorizadas en BT y EU

La opción SUBTYPE=TYPE se utiliza cuando se compila el benchmark de E/S BT, pudiendo definir los tipos en: full, simple, fortran o epio.

La ejecución de la instrucción make, construye un benchmark de manera individual, ubicando el programa resultante en la carpeta bin. Se puede compilar toda la suite con el comando make suite, donde en este caso, make buscará en el archivo config/suite.def una lista de archivos para construir.

Todos los benchmarks están diseñados para poderse utilizar en un único procesador sin la librería MPI. Algunas de las rutinas requeridas en el proceso del linker pueden ser completadas con la librería dummy facilitada en el benchmark, esta librería se define también en el fichero make.def.

Un ejemplo de compilación podría ser el siguiente:

```
[root@cluster NPB3.3-MPI]# make is NPROCS=1 CLASS=A
```

Este comando generaría el binario correspondiente al benchmark IS - Integer Sort, para 1 proceso, con un tamaño de problema de clase A.

El ejecutable obtiene un nombre siguiendo el siguiente patrón:

<nombre-benchmark>.<clase>.<núm procesos>[.<sufijo>]

Este binario estará ubicado en la carpeta bin del subdirectorio o en la ruta que se haya definido en el fichero make.def en la variable BINDIR. El método para ejecutar el programa MPI dependerá del sistema, donde en el caso expuesto la sintaxis correcta de ejecución sería la misma que la utilizada anteriormente para lanzar programas MPI, pudiendo englobarlo dentro de time para obtener una medida aproximada del tiempo de ejecución empleado por el programa en completarse.

4.9.7 Ejecución de benchmarks

A continuación se explicará el procedimiento de ejecución de los diferentes benchmarks que componen la suite NAS Parallel Benchmark.

Se ejecutarán los diferentes binarios obtenidos de la compilación anterior, en orden creciente según la carga de trabajo que supongan para el sistema, comenzando con la clase más mínima definida (W) y ejecutando gradualmente las clases superiores desde 1 nodo a los máximos implicados en la ejecución.

Para simplificar el proceso de lanzamiento de ejecuciones con modificación de parámetros de clase o número de nodos se han desarrollado unos scripts que facilitan esta tarea, ya que aprovechan el sistema de colas de trabajos proporcionado en el sistema cluster Sun Grid Engine, util para la ejecución del benchmark indicado junto a dos parámetros que definirán su carga y número de nodos.

A continuación se puede observar el código de un script utilizado para la ejecución:

```
#!/bin/bash
#$ -cwd
#$ -j y
#$ -S /bin/bash
#
#MPI_DIR=/opt/mpich/gnu/
#HPL_DIR=/opt/hpl/mpich-hpl/

#Parametro 1: Número de procesos
$1

#Parametro 2: Classe
$2

mpirun -np $1 -machinefile ../machines/mach$1 ../bin/CG/CLASS_$2/cg.$2.$1 >> CG/cg.$2.
    $1.out
```

En el código anterior se detalla un script para el lanzamiento del benchmark CG, donde se hubiera podido simplificar aún más pasando también por parámetro las iniciales del benchmark simplificado, pero para mantener la seguridad se ha mantenido el paso de parámetros solo para la carga de clase y las máquinas implicadas. La finalización de esta tarea pone en un fichero de texto la salida en pantalla de la ejecución del programa.

La sintaxis de la ejecución para encolar un trabajo en el sistema GridEngine sería el siguiente:

```
[zxcv90@cluster scriptsGrid]#qsub grid_CG.sh 4 B
```

El primer parámetro **4**, determina el número de procesos máquina que se lanzarán durante la ejecución, el segundo parámetro **B**, determina la clase de la suite con la que se intentará ejecutar. Comentar que previamente los programas deben de haberse compilado, estos scripts no realizan dicha tarea.

Tras la ejecución correcta del programa se generarán ficheros de salida en el path que se haya determinado en la última línea del script, en este caso **CG/cg.$2.$1.out**, manteniendo el formato en el nombre de clase y número de procesos para facilitar su clasificación posterior.

Dentro de los fichero guardados se puede visualizar la información de salida de diferentes métricas y datos importantes que ha generado el programa, toda la información se puede encontrar al final del archivo en caso de finalizar correctamente. A continuación se muestra un ejemplo:

```
 MG Benchmark Completed.
Class           =                        A
Size            =           256x 256x 256
Iterations      =                        4
Time in seconds =                     8.07
Total processes =                        4
Compiled procs  =                        4
Mop/s total     =                   482.22
Mop/s/process   =                   120.55
Operation type  =             floating point
Verification    =                SUCCESSFUL
Version         =                      3.3.1
Compile date    =              29 Oct 2016
```

4.10 NAS Parallel Benchmarks: Running basic Kernels

En el siguiente apartado se mostrarán los resultados de tiempo total de ejecución sobre los diferentes kernels básicos de la suite NAS (IS,EP,CG,MG,FT).

Las ejecuciones de los programas se han lanzado variando la carga de cálculo, adecuando las clases disponibles de configuración para cada kernel. Algunas cargas en kernels puntuales no se han podido ejecutar en los computadores del sistema cluster implementado por límite de recursos cuando se despliega el programa, uno de estos recursos afectados es principalmente la memoria RAM.

En los siguientes resultados se puede observar una tabla con los diferentes tiempos de ejecución del programa con varias cargas de trabajo, ejecutadas en diferentes nodos del sistema, definiendo su máximo en 4 nodos. También se adjuntan gráficas en cada apartado, representando el tiempo de ejecución en formato de histograma y a continuación el speedup aproximado de tiempo de ejecución entre diferentes configuraciones.

4.10.1 Kernel IS: Resultados

Clase	1 Nodo	2 Nodos	4 Nodos
W	0,61	1,27	1,26
A	4,97	10,6	9,68
B	19,95	42,03	36,15
C	-	168,83	143,73

Tabla 4.14: Tiempos de ejecución en Kernel IS

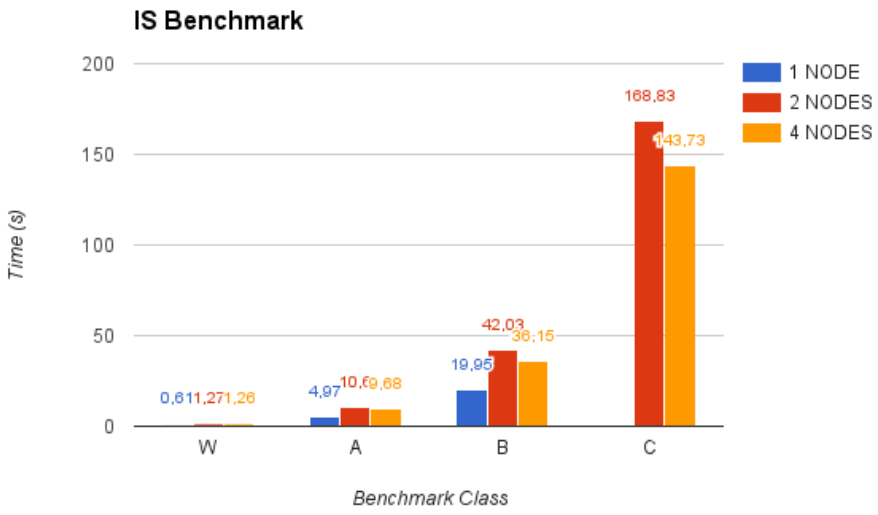


Figura 4.9: Gráfica de barras con tiempos de ejecución en Kernel IS

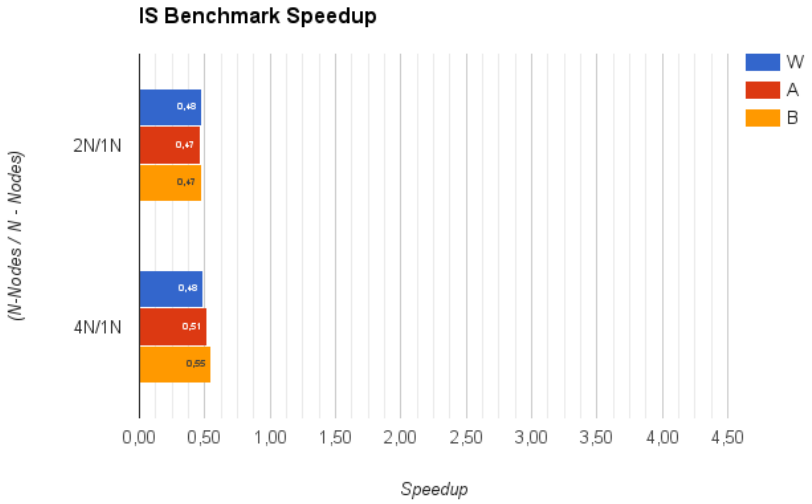


Figura 4.10: Speedup en Kernel IS

4.10.2 Kernel EP: Resultados

Clase	1 Nodo	2 Nodos	3 Nodos	4 Nodos
W	8,88	4,43	2,95	2,22
A	70,63	35,51	23,58	17,77
B	282,73	141,36	94,63	70,69
C	1139,56	569,30	377,22	282,73

Tabla 4.15: Tiempos de ejecución en Kernel EP

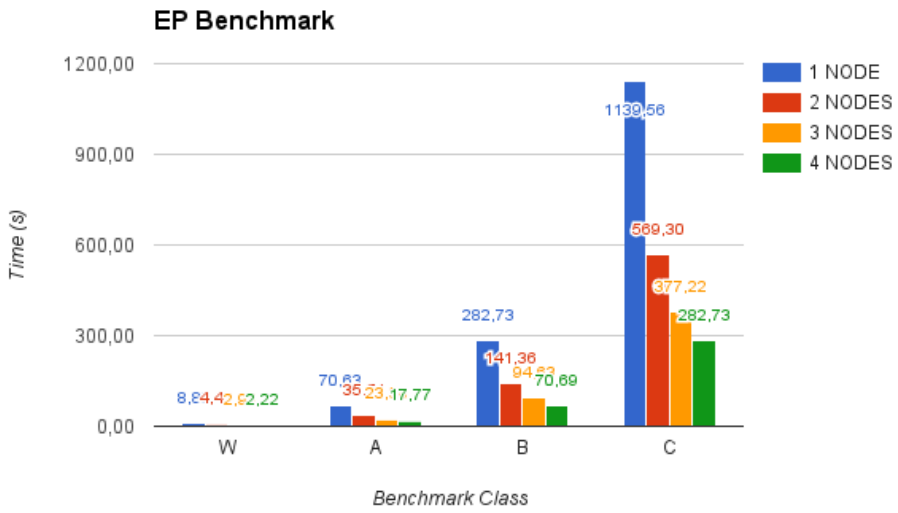


Figura 4.11: Gráfica con tiempos de ejecución en Kernel EP

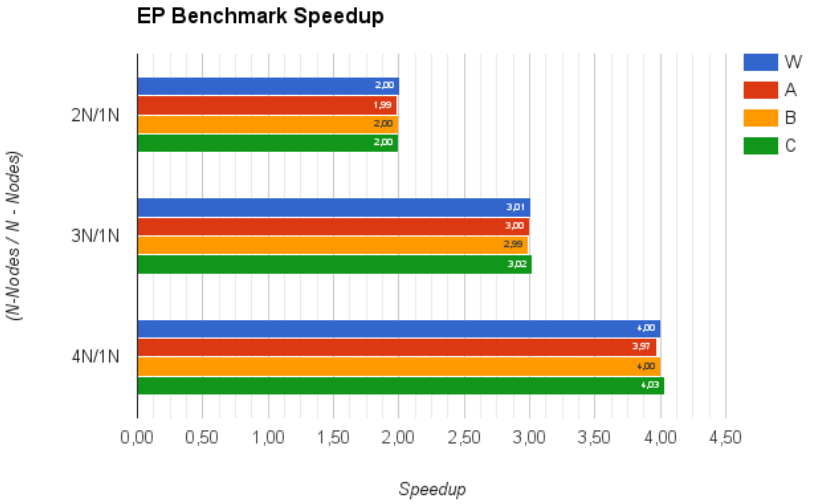


Figura 4.12: Speedup en Kernel EP

4.10.3 Kernel CG: Resultados

Clase	1 Nodo	2 Nodos	4 Nodos	2 Nodos P4 775
W	2,52	2,44	3,09	-
A	8,92	6,7	7,49	-
B	1087,55	268,67	261,08	192,07

Tabla 4.16: Tiempos de ejecución en Kernel CG

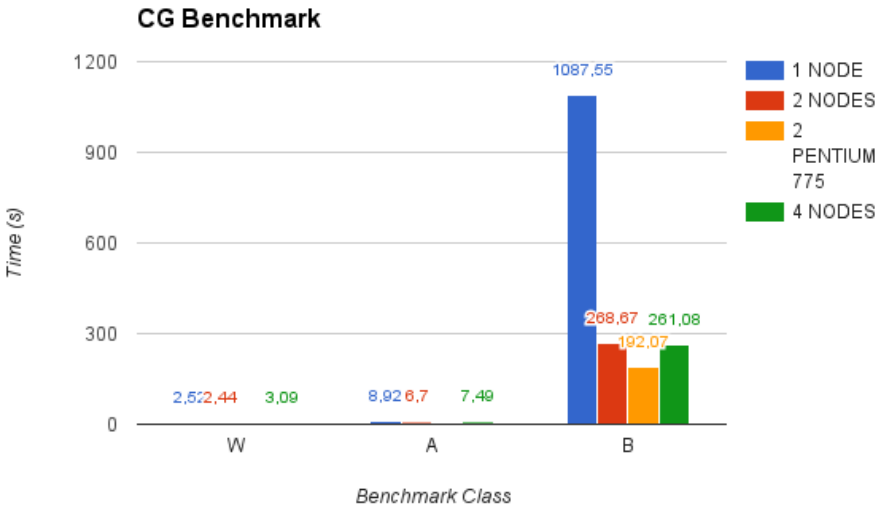


Figura 4.13: Gráfica con tiempos de ejecución en Kernel CG

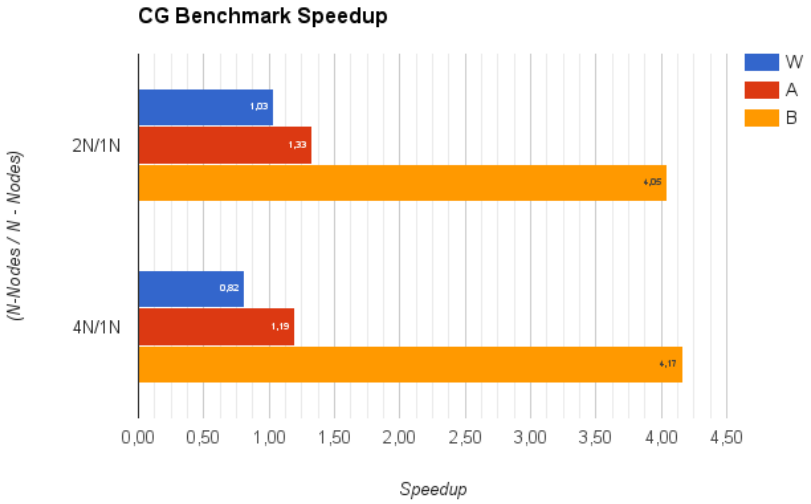


Figura 4.14: Speedup en Kernel CG

4.10.4 Kernel MG: Resultados

Clase	1 Nodo	2 Nodos	4 Nodos
W	3,2	1,91	1,34
A	21,64	12,5	8,07
B	101,14	59,61	37,09
C	-	-	239,84

Tabla 4.17: Tiempos de ejecución en Kernel MG

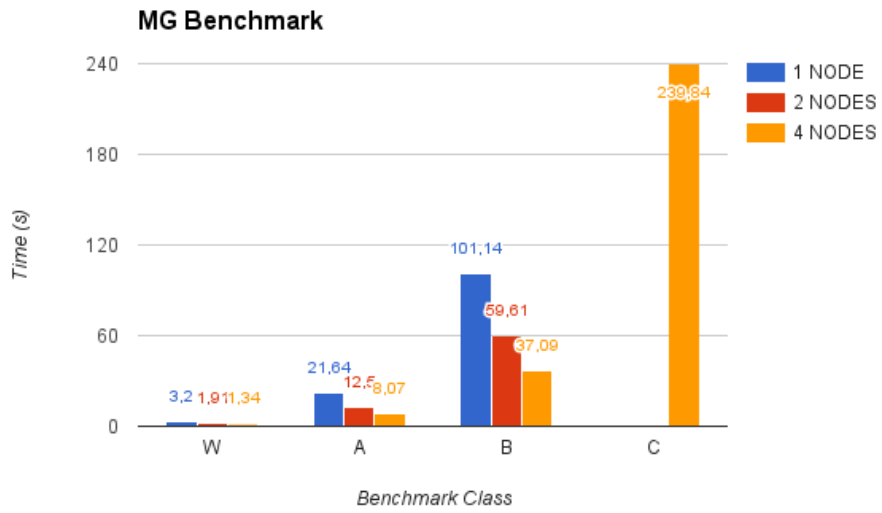


Figura 4.15: Gráfica con tiempos de ejecución en Kernel MG

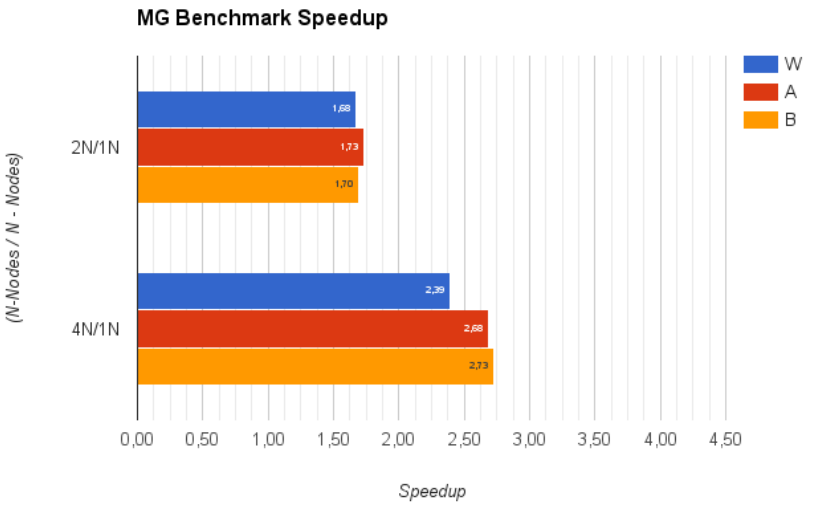


Figura 4.16: Speedup en Kernel MG

4.10.5 Kernel FT: Resultados

Clase	1 Nodo	2 Nodo	4 Nodo
W	2,3	2,71	2,05
A	40,62	44,51	33,45
B	-	530,98	400,13

Tabla 4.18: Tiempos de ejecución en Kernel FT

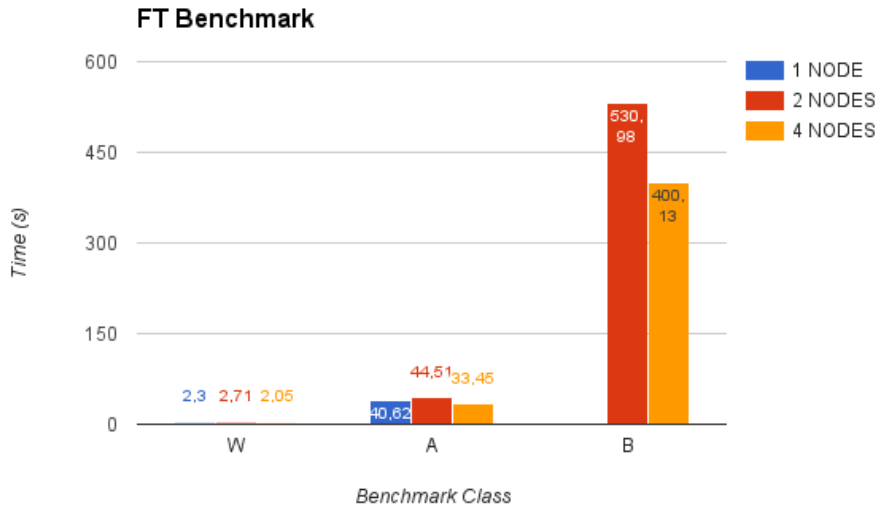


Figura 4.17: Gráfica con tiempos de ejecución en Kernel FT

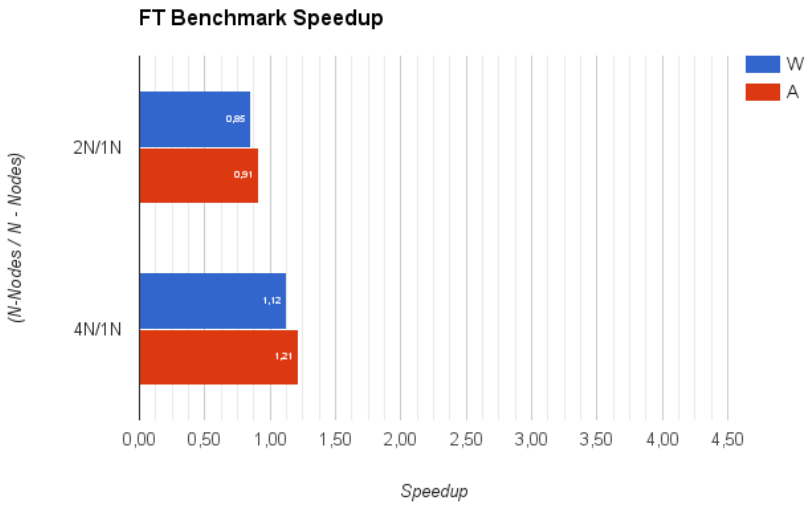


Figura 4.18: Speedup en Kernel FT

4.10.6 Conclusiones ejecución Kernels

La ejecución de los diferentes kernels simples de la suite NAS muestran algunos fenómenos interesantes aplicados sobre la configuración actual del sistema.

En las distintas ejecuciones, comúnmente, la inclusión de más máquinas de cálculo incrementa el rendimiento del conjunto reduciendo en mayor o menor medida el tiempo de ejecución cuando están implicados más nodos en la resolución del problema. También se puede destacar, que los kernels que requieren de mayor recurso de sincronización a través de la red, el rango de mejora es más limitado.

En los Kernels que ejecutan operaciones de coma flotante, el valor máximo de millones de instrucciones en coma flotante por segundo se muestra a continuación, este se ha obtenido en el Kernel MG - Clase C lanzado sobre 4 nodos.

```
Compiled procs  =                 4
Mop/s total     =            649.16
Mop/s/process   =            162.29
Operation type  =      floating point
Verification    =        SUCCESSFUL
```

MFLOPS= 649.16

Durante la ejecución del **Kernel IS**, la ejecución de las clases de menos carga computacional W,A,B, se alcanzan unos valores bajos de speedup, donde los tiempos de ejecución son más rápidos ejecutando el kernel sobre solo una máquina. No obstante, el incremento de clase y el tamaño del problema, **desbordan la ejecución cuando se utiliza solo un nodo**, limitando su ejecución a partir de 2 máquinas, debido a la limitación de RAM en los equipos de manera individual.

En la ejecución del **Kernel EP**, siendo el único que permite la ejecución en N máquinas indiferentemente de que sean potencias de 2, se alcanzan **unos de los valores más altos de speedup, en torno a (4.00 aprox) puntos de mejora** ejecutando el kernel en las 4 máquinas disponibles frente a la ejecución en solo 1 máquina. Este incremento de mejora se mantiene progresivo con la inclusión desde 1 a 4 máquinas tal como puede observarse en las gráficas, se trata de un incremento ideal, pues el algoritmo ejecutado basa su funcionamiento en distribuir la carga de trabajo entre las máquinas disponibles donde solo interviene la comunicación de sincronización en la última fase, evitando que la utilización de la red degrade el tiempo de ejecución final del programa.

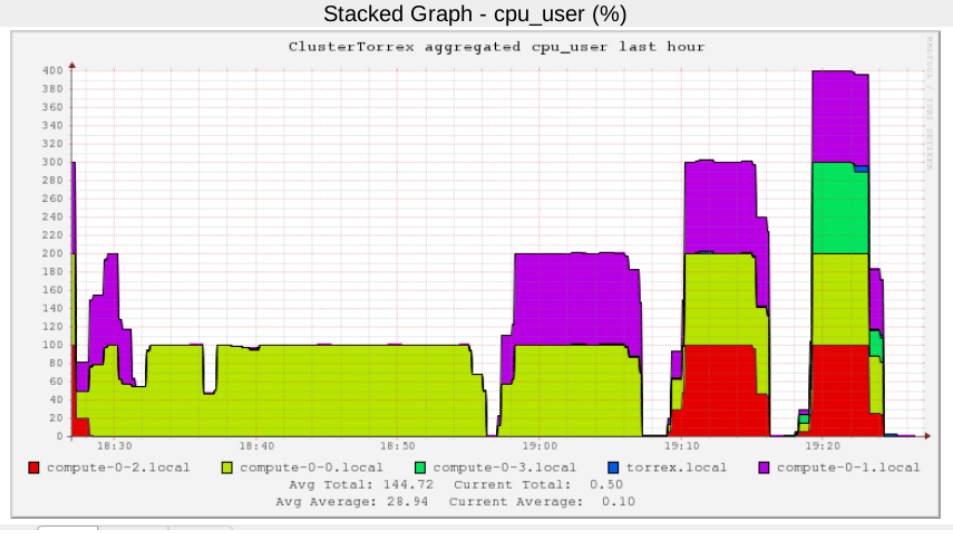


Figura 4.19: Captura de monitor Ganglia con el porcentaje de uso de procesador sobre línea de tiempo en la ejecución del Kernel EP clase C en 1,2,3 y 4 nodos.

En la ejecución del **Kernel CG**, se puede observar que los tiempos de ejecución se mantienen similares en las clases inferiores (W,A) indiferentemente de las máquinas implicadas en la ejecución, debido a que las iteraciones que realizan estas clases se mantienen en torno a 15 loops, con unos parámetros de entrada similares. Es en la ejecución de una clase superior (B) donde las iteraciones del algoritmo ascienden a 75 loops, y donde se observa la diferencia en tiempo de ejecución final entre 1 máquina y 2 o 4. Destacar, que los tiempos entre 2 o 4 máquinas mantienen un tiempo similar (268,67s - 261,08s), fenómeno que puede delatar que el algoritmo precisa de una alta utilización de la red de interconexión, no siendo eficiente la fragmentación del problema en todas las máquinas que se disponen en el cluster implementado sin disponer de una alta velocidad de interconexión como es el caso.

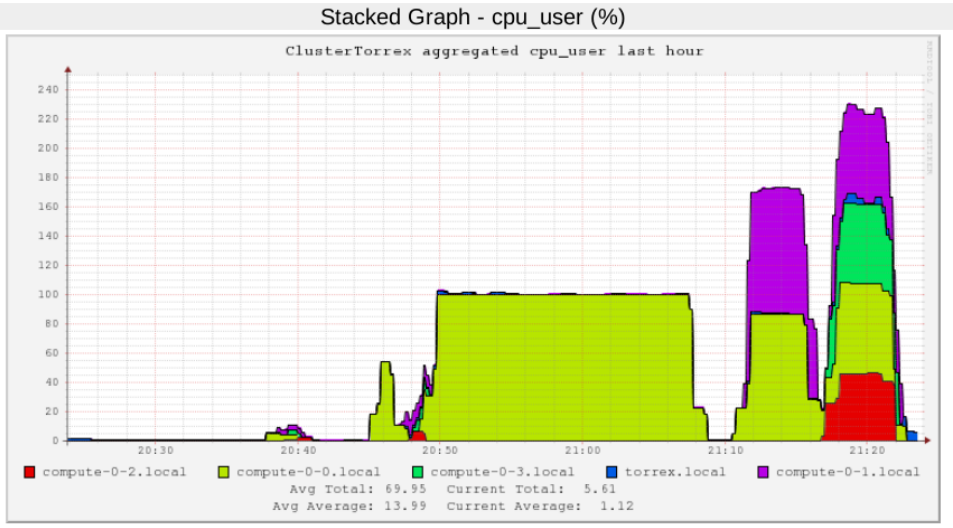


Figura 4.20: Captura de monitor Ganglia con el porcentaje de uso de procesador sobre línea de tiempo en la ejecución del Kernel CG clase B en 1,2 y 4 nodos.

Como extra, se realiza la ejecución del Kernel CG solo en las **2 máquinas que montan Pentium 4 620**. Con esta configuración **se reduce el tiempo final a (192.07s)**, obteniendo una mejora sobre una máquina de un **speedup: 5.71 puntos**.

De este último resultado se podría concluir que limitarse a la inclusión de más números de procesadores no siempre obtendrá el mejor ratio de mejora, siendo posible conseguir mejor rendimiento con la distribución adecuada de la carga de trabajo del problema o la utilización de procesadores más potentes pero en menor número, siempre dependiendo de la necesidad de la problemática planteada.

En la ejecución del **Kernel MG**, se podría destacar que en la ejecución de las clases iniciales (W, A, B), se mantiene un incremento en el rendimiento siguiendo una línea ideal, logrando valores de speedup de 1.70 puntos con ejecución en 2 máquinas y 2.5 aproximadamente en la ejecución en 4 máquinas. Con el incremento del tamaño de problema a la **clase C**, el kernel solo es ejecutable en 4 nodos, principalmente por la **falta de recursos de memoria RAM** sí se trata de ejecutar en un número menor de máquinas. Aún así, es el Kernel que consigue una optimización para el alcance del mayor número de operaciones en coma flotante por segundo: **649.16 MFLOPS**

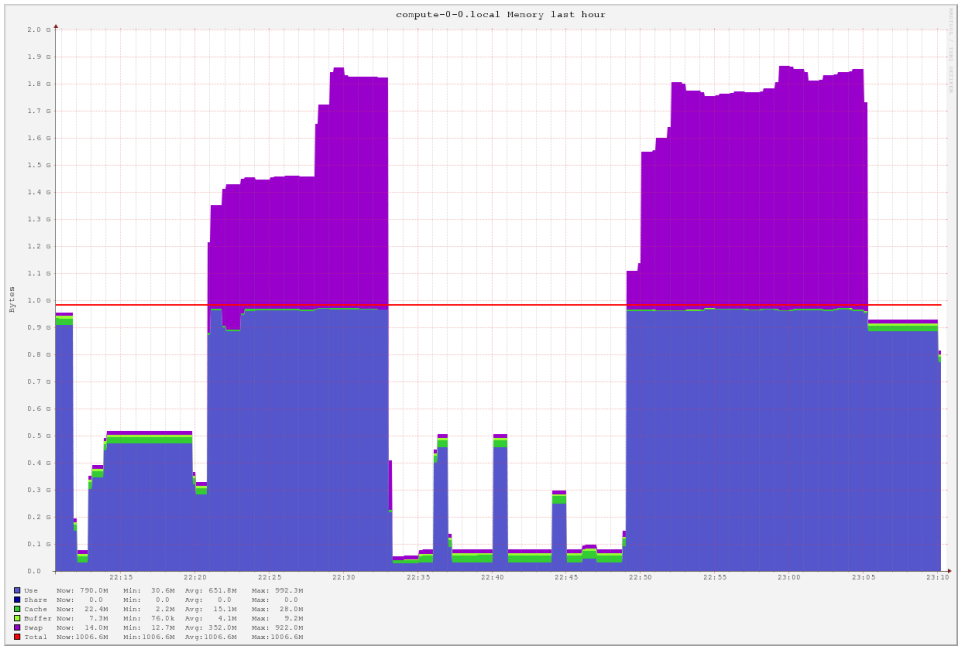


Figura 4.21: Sobrecarga de memoria en la ejecución del Kernel MG clase C en 1 nodo del cluster. El color morado representa la utilización de memoria SWAP del sistema, trabajando directamente sobre el disco y degradando drásticamente el rendimiento de la ejecución hasta no poder ejecutar fluidamente el Kernel.

En la ejecución del **Kernel FT**, en las clases (W, A) el tiempo de ejecución es similar indiferentemente de la ejecución en mayor o menor número de nodos. Sí se incrementa el tamaño de problema a clase B, la ejecución en un nodo deja de poder realizarse por falta de recursos en memoria RAM, limitando la ejecución a 2 i 4 nodos, donde el ratio de mejora entre la ejecución en 4/2 nodos arroja un valor de speedup de 1.33 puntos.

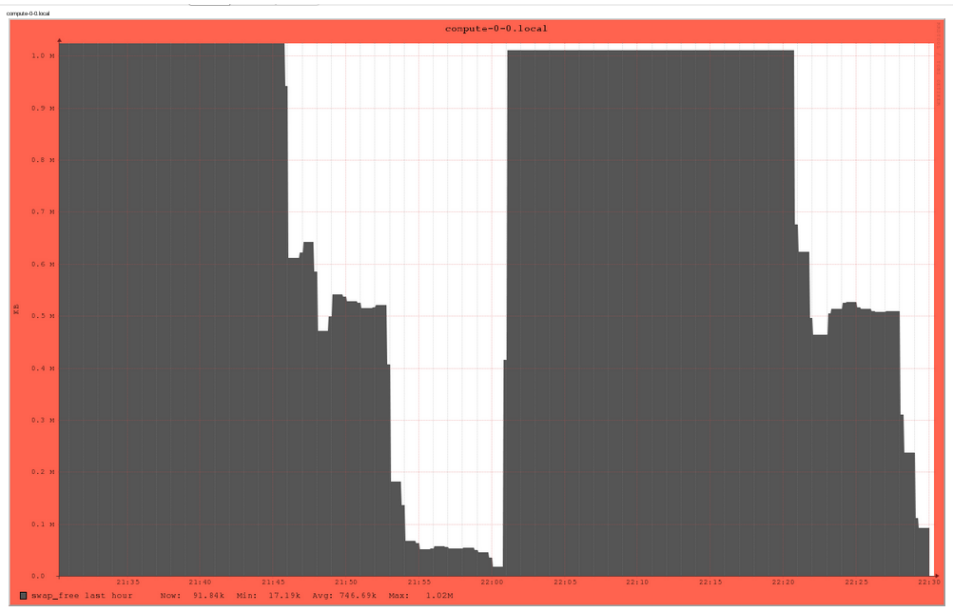


Figura 4.22: Captura del monitor de Ganglia sobre uso de memoria SWAP libre (espacio oscuro). Se puede observar que el lanzamiento del Kernel FT clase B consume la SWAP, utilizando recursos de disco por limitación de RAM y empeorando drásticamente el rendimiento.

4.11 CFLAGS/FFLAGS: optimización compiladores

Las variables de entorno CFLAGS y CXXFLAGS son las que se utilizan convencionalmente para especificar opciones de compilación en un sistema de construcción cuando se compila código C, C++ o Fortran. Aunque estas variables no están estandarizadas, su utilización es esencialmente ubicua y cualquier construcción escrita correctamente debería interpretarlas de forma adecuada para el paso de opciones extra o personalizadas cuando se invoca el compilador.

La utilización de estas opciones, están orientadas a la optimización de la compilación, donde el hecho de activar algunas puede resultar muy efectivo a la hora de producir binarios, aunque también es necesario mencionar que algunas configuraciones aplicadas pueden deteriorar el rendimiento del código, inflando el tamaño del binario final o ralentizando el tiempo de ejecución

Para su uso, es necesario definir las variables CFLAGS, normalmente se debe de modificar en la invocación de un guión o ficheros makefile, o paso como parámetro en el comando de compilación del binario. En el caso de la suite de NAS Benchmarks, estas variables se pueden definir en el fichero config/make.def, por ejemplo en la siguiente línea para el compilador de C:

```
#---------------------------------------------------------------------------
# Global *compile time* flags for C programs
#---------------------------------------------------------------------------
CFLAGS =
```

En estas variables de entorno se deben de incluir los modificables que afectarán a la compilación del programa final. A continuación se detallan algunas de las opciones posibles para tratar de mejorar el rendimiento del código fuente de un programa.

4.11.1 Objetivos:

El objetivo de esta sección es utilizar los flags de mejora disponibles en los compiladores de C y Fortran77 para verificar sí su utilización ofrece una mejora de rendimiento y ganancia de speedup en la ejecución de un kernel de la suite NAS sobre el cluster implementado, mostrando otra manera de conseguir más rendimiento.

4.11.2 Variable de optimización

El objetivo principal de CFLAGS o FFLAGS en los compiladores gcc (código C) y mpif77 (Fortran77) es crear código específico para el sistema donde se ejecutará el programa, aplicando una serie de mejoras en la generación de binarios durante el tiempo de compilación. Para que un programa sea eficiente debería de ser ligero, rápido y funcionar correctamente, pero estas condiciones a veces son excluyentes, donde de manera ideal, las mejores optimizaciones están disponibles para cada arquitectura de CPU. A continuación se detallan algunas de las opciones.

- **-march** La primera opción es -march. Esta opción le indica al compilador qué código se debería de producir para una arquitectura de procesador concreta. Diferentes CPU tienen diferentes características y soportan diferentes conjuntos de instrucciones. La opción -march indicará al compilador que produzca el código específico para la CPU del sistema tomando en cuenta todas sus capacidades, características, juegos de instrucciones y demás.

  Sí no se puede determinar el tipo de CPU es posible utilizar el ajuste -march=native, donde al utilizarla el compilador trataría de buscar automáticamente las opciones apropiadas. Esto no se debería de utilizar en el caso de que el sistema incluya CPUS diferentes ya que esto solo compila el código para la arquitectura de la máquina origen.

En el caso expuesto, donde los procesadores de los nodos son todos modelos Intel Pentium 4, se debería de configurar la opción de la siguiente manera:

```
CFLAGS = "-march=pentium4"
```

- **-O** La variable -O controla el nivel de optimización de todo el código. Al cambiar este valor, la compilación de código tomará más tiempo, y utilizará más memoria, pero incrementará el valor de optimización.

  Existen ajustes para -O: -O0,-O1,-O2,-O3,-Os,-Og y -Ofast.

  Se debe de utilizar solo uno de ellos.

  Se indicará ahora el nivel de optimización de cada uno de ellos:

  - **-O0** Este nivel desconecta por completo la optimización.
  - **-O1** Nivel de optimización básico. El compilador intentará producir un código rápido y pequeño sin tomar mucho tiempo de compilación.
  - **-O2** Nivel superior de -01. Es el nivel recomendado de optimización, a no ser que el sistema tenga necesidades especiales. Este nivel activará algunas opciones añadidas, intentado aumentar el rendimiento del código sin comprometer el tamaño.
  - **-O3** Nivel más alto de compilación posible. Activa opciones que son caras en términos de tiempo de compilación y uso de memoria. El hecho de compilar con -03 no garantiza una forma de mejorar el rendimiento, de hecho, en algunos casos puede ralentizar el sistema debido al uso de binarios de gran tamaño y mayor uso de memoria. El uso de -03 no es el más recomendable.
  - **-Os** Optimizará el tamaño del código. Activa todas las opciones de -02 que no incrementan el tamaño del código generado. Útil para máquinas con capacidad limitada de disco o CPU con poca cache.
  - **-Og** Trata de solucionar la necesidad de realizar compilaciones más rápidas y obtener una experiencia superior en la depuración a la vez que ofrece un nivel razonable de rendimiento en la ejecución. Este nivel debería ser mejor que -O0.
  - **-Ofast** Consiste en el ajuste -03 más las opciones -ffast-math, -fno-protect-parens y -fstack-arrays. Esta opción rompe el cumplimiento de estándares y no se recomienda su utilización.

    Como se cita anteriormente, -02 es el nivel de optimización recomendado.

- **-pipe** No tiene efecto sobre el código que produce, pero hace que el proceso de compilación sea más rápido.

A continuación se procederá a ejecutar un kernel de la suite con diferentes niveles de optimización en el proceso de compilación para ver el impacto que tiene el uso de estas opciones en el rendimiento final del programa, tanto a nivel individual en una máquina como en conjunto en ejecución sobre el cluster.

El kernel seleccionado para probar la optimización del código es el: **EP BENCHMARK**, configurado en un tamaño de problema de **clase B**.

Se probarán las siguientes opciones de optimización: -march, -O1, -O2, -O3

Los resultados originales obtenidos en la ejecución de este benchmark fueron los siguientes:

Clase	1 Nodo	2 Nodos	3 Nodos	4 Nodos
B	282,73	141,36	94,63	70,69

Tabla 4.19: Tabla tiempos de ejecución original Kernel EP

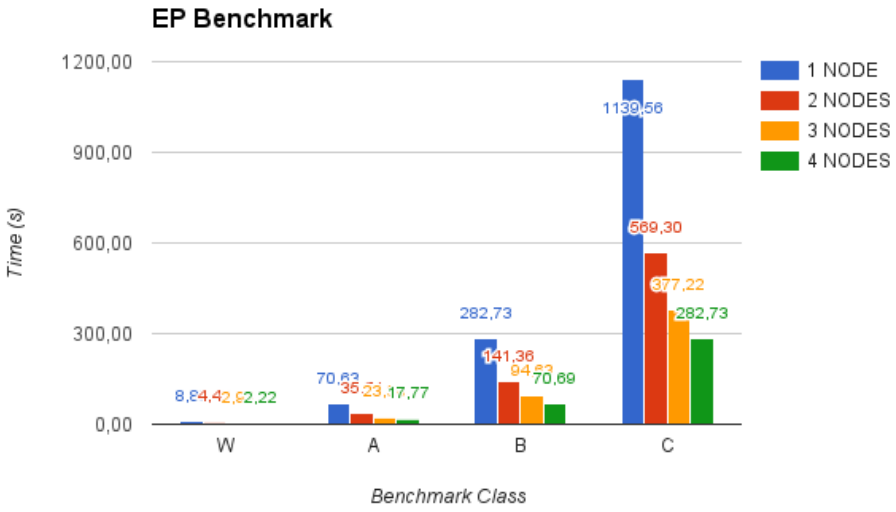


Figura 4.23: Gráfica tiempos de ejecución Kernel EP

4.11.3 CFLAG: -march='pentium4'

En el fichero de configuración make.def se define la variable CFLAGS con la siguiente cadena:

```
CFLAGS = "-march=pentium4"
```

Se procede a la compilación, ejecución y medida de tiempo correspondiente para el kernel de la suite de benchmarks.

Los resultados de tiempo son los siguientes:

Clase	1 Nodo	2 Nodos	3 Nodos	4 Nodos
B	289,14	145,12	96,67	72,33

Tabla 4.20: Tabla tiempos de ejecución Kernel EP (s) con CFLAG: -march='pentium4'

4.11.4 CFLAG: -01

En el fichero de configuración make.def se define la variable CFLAGS con la siguiente cadena:

```
CFLAGS = "-O1"
```

Se procede a la compilación, ejecución y medida de tiempo correspondiente para el kernel de la suite de benchmarks.

Los resultados de tiempo son los siguientes:

Clase	1 Nodo	2 Nodos	3 Nodos	4 Nodos
B	251,37	125,64	83,76	62,80

Tabla 4.21: Tabla tiempos de ejecución Kernel EP (s) con CFLAG: '-O1'

4.11.5 CFLAG: -02

En el fichero de configuración make.def se define la variable CFLAGS con la siguiente cadena:

```
CFLAGS = "-O2"
```

Se procede a la compilación, ejecución y medida de tiempo correspondiente para el kernel de la suite de benchmarks.

Los resultados de tiempo son los siguientes:

Clase	1 Nodo	2 Nodos	3 Nodos	4 Nodos
B	241,44	241,44	80,87	60,35

Tabla 4.22: Tabla tiempos de ejecución Kernel EP (s) con CFLAG: '-O2'

4.11.6 CFLAG: -03

En el fichero de configuración make.def se define la variable CFLAGS con la siguiente cadena:

```
CFLAGS = "-O3"
```

Se procede a la compilación, ejecución y medida de tiempo correspondiente para el kernel de la suite de benchmarks.

Los resultados de tiempo son los siguientes:

Clase	1 Nodo	2 Nodos	3 Nodos	4 Nodos
B	240,84	121,06	80,33	60,26

Tabla 4.23: Tabla tiempos de ejecución Kernel EP (s) con CFLAG: '-O3'

4.11.7 Sumario tiempos de ejecución y speedup

Como se puede comprobar, la modificación de las flags pasadas en el proceso de precompilación, modifican los tiempos finales de ejecución del kernel, mejorando en algunos casos de manera significativa el rendimiento de la aplicación. A continuación se adjunta una tabla con el sumario de todos los tiempos obtenidos con los diferentes flags utilizados, además de unas gráficas para apreciar las diferencias de tiempos, junto el incremento de mejora en valores de speedup.

CFLAG	1 Nodo	2 Nodos	3 Nodos	4 Nodos
Base	282,73	141,36	94,63	70,69
-march: "pentium4"	289,14	145,12	96,67	72,33
-O1	251,37	125,64	83,76	62,80
-O2	241,44	241,44	80,87	60,35
-O3	240,84	121,06	80,33	60,26

Tabla 4.24: Tabla tiempos de ejecución Kernel EP (s) con diferentes valores de CFLAGS

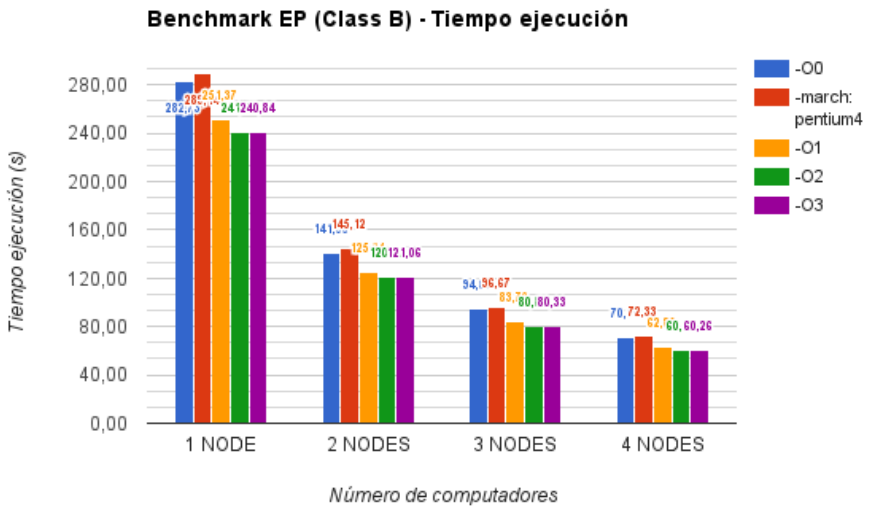


Figura 4.24: Tiempos de ejecución Kernel EP (s) con diferentes valores de CFLAGS

Como puede observarse, utilizando valores CFLAGS de optimización más agresivos como -O2 o -O3, se obtiene un mejor tiempo de ejecución en el programa, no dándose el mismo caso en el flag de optimización por procesador -march="pentium4", donde el rendimiento empeora comparado con el tiempo original de ejecución del kernel.

Con los tiempos de ejecución obtenidos, es posible calcular los valores de speedup que se necesiten comparando los rendimientos que se crean más relevantes. En el caso expuesto, se obtendrán los valores de speedup para cada flag diferenciándolos en el número de computadores implicados en la ejecución del programa, este valor será la comparativa del tiempo obtenido con la optimización realizada en el compilador, contra el tiempo de ejecución estándar del kernel. Los valores de speedup se detallan en la tabla y gráfica siguiente.

CFLAG	1 Nodo	2 Nodos	3 Nodos	4 Nodos
-march: "pentium4"	0,98	0,97	0,98	0,98
-O1	0,98	0,97	0,98	0,98
-O2	1,12	1,13	1,13	1,13
-O3	1,17	1,17	1,18	1,17

Tabla 4.25: Speedup de la optimización por CFLAGS entre distintos nodos de ejecución comparados con el tiempo original.

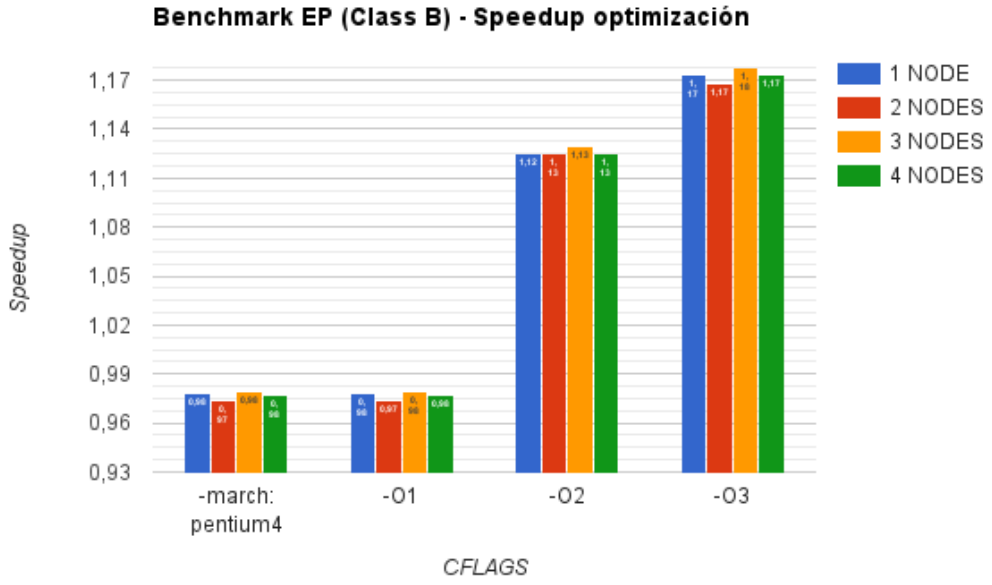


Figura 4.25: Grafica de Speedup obtenido en diferentes CFLAGS

4.11.8 Conclusión

Como se puede observar en los datos obtenidos anteriormente. El uso de la optimización disponible en los compiladores mpif77 y gcc, permite mejorar mínimamente los tiempos finales de ejecución de un programa. Añadiendo esta pequeña mejora al incremento de rendimiento obtenido por la inclusión de diferentes máquinas de cómputo para la resolución de algunos problemas, podemos determinar la mejora total obtenida en relación al tiempo de ejecución total en una sola máquina.

Para representar esto, se realizará la comparativa entre el tiempo de ejecución del Kernel EP en una sola máquina de cómputo, sin el uso de optimización en la precompilación del código, y el tiempo de ejecución en 4 nodos aplicando la mejora mediante CFLAGS. Los valores de tiempo para estas dos configuraciones serían las siguientes:

Optimización	Tiempo ejecución (s)
-	282,73
CFLAGS='-O3'	60,26

Tabla 4.26: Tiempo de ejecución en las dos configuraciones (ejecucion estandar y ejecucion en 4 nodos con optimización -03).

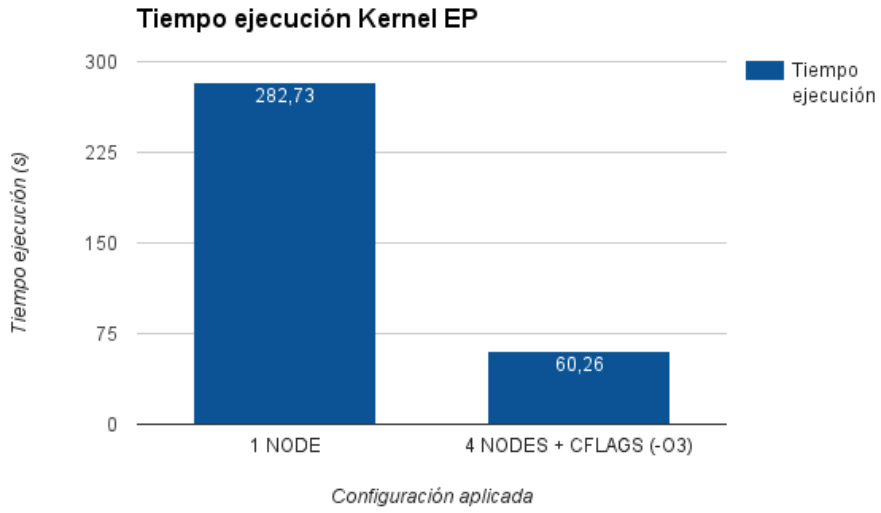


Figura 4.26: Tiempo de ejecución en las dos configuraciones

Como se puede apreciar, la mejora final del tiempo de ejecución del Kernel EP es bastante significativa, ahorrando mucho tiempo para la resolución final del programa. Sí se calcula el valor del speedup entre estos dos tiempos, se obtiene un valor de:

Speedup= 4,69

Con este valor se puede determinar un incremento de rendimiento total de **x4,7** veces más rápido sumando las condiciones de mejora tanto a nivel de código con la utilización de optimización en tiempo de compilación como en el incremento de máquinas de cómputo para paralelizar el cálculo del problema. **El valor de speedup es 0.69 puntos superior al valor de speedup original en ejecución paralela sobre 4 nodos.**

Es necesario indicar, que estas modificaciones afectan de manera positiva al rendimiento en este caso, sobre el Kernel EP de la suite NAS Benchmark, pero esta mejora no es implícita a cualquier ejecución, pudiendo verse afectado el rendimiento de forma negativa en otras configuraciones que no sean acordes a una problematica en concreto.

Capítulo 5

Cluster profiling: BSC-Tools

5.1 Introducción

En la complejidad de la programación de aplicaciones sobre arquitecturas avanzadas de computadores, multicomputadores, clusters, etc, suele ser necesario el uso de herramientas de análisis de rendimiento enfocadas a obtener información relevante para el desarrollador que permitan comprender con más detalle aspectos en la eficiencia y rendimiento de la aplicación desarrollada.

Dentro del modelo de ejecución paralela donde se busque el rendimiento y la eficiencia, este es un paso importante para lograr un análisis con más profundidad y determinar posibles causas de una mala optimización. Es por ello que, algunas instituciones dedicadas a la investigación, han desarrollado sus propias herramientas software enfocadas en el análisis de rendimiento sobre plataformas basadas en paralelismo.

Una de estas instituciones, dentro del territorio nacional es el centro de supercomputación de Barcelona (BSC - Barcelona Supercomputing Center). En esta institución se empezó a desarrollar un conjunto de herramientas enfocadas al análisis de rendimiento sobre plataformas paralelas, este conjunto de herramientas se engloban dentro del conjunto llamado CEPBA-Tools/BSC-TOOLS toolkit, que incluyen programas para el análisis de la ejecución paralela, disponibles para diferentes modelos y lenguajes de programación. En resumen:

- **Paraver:** Se trata de la herramienta de análisis y visualización de trazas generadas. Permite múltiples visualizaciones y diferentes vistas de información contenida en los archivos de trazas generadas con un alto nivel de configuración. Se detallará más adelante.
- **Dimemas:** Herramienta de simulación para el análisis paramétrico del comportamiento de aplicaciones orientadas al paso de mensajes en una plataforma paralela. Es un simulador de MPI que aproxima la predicción de la ejecución de una aplicación en una máquina abstracta definiendo previamente valores de red y de CPU. Está capacitada para generar una traza para Paraver que represente la ejecución aproximada con los parámetros que el usuario ha definido.
- **Extrae:** El paquete de instrumentación desarrollado en el kit de Performance Tools del BSC. Utilizado para generar trazas Paraver. Basado en la instrumentación del código en su ejecución, está capacitado para la instrumentación de aplicaciones basadas en distintos modelos de programación (OpenMP, MPI, CUDA, etc) y usar diferentes aproximaciones

de instrumentaciones. La información generada por Extrae típicamente incluye eventos de tiempo y llamadas de runtime, contadores de rendimiento y referencias a código fuente.

Estas herramientas son con licencia free-software y están disponibles para su descarga en la página oficial del centro:

`https://tools.bsc.es/downloads`

5.2 Paraver

Paraver es un programa para la visualización del flujo de ejecución de programas paralelos y una herramienta de análisis basada en el interface Motif GUI. Fue desarrollada para suplir la necesidad de disponer de una herramienta que aportara una percepción general del comportamiento de una aplicación de forma visual, facilitando el análisis de detalles cuantitativos y el análisis de problemas.

Las características de Paraver ofrecen soporte para:

- Análisis detallado del rendimiento de programas.
- Análisis comparativo entre diferentes trazas de ejecución.
- Capacidad de guardar las visualizaciones para uso posterior.
- Desarrollo de métricas derivadas.

Está basado en el análisis directo de trazas de ejecución generadas por otros programas del toolkit del BSC-Tools, como Dimemas o Extrae. Principalmente, el análisis de trazas no tiene semántica, por lo que facilita que este formato este disponible para cualquier modelo de programación nuevo sin requerir cambios en el visualizador Paraver.

Algunos de los modelos de programación soportados son:

- MPI
- OpenMP
- phtreads
- OmpSs
- CUDA

El visualizador Paraver, dispone de múltiples vistas altamente configurables para una amplia gama de métricas directamente generadas de la ejecución de programas. La información se muestra en un timeline que representa el comportamiento de las aplicaciones a través del tiempo y procesos, mostrando diferentes fases, patrones y eventos, lo que simplifica el hecho de entender el comportamiento de la aplicación durante su ejecución en el sistema. También ofrece configuraciones de estadísticas, configurables para obtener datos relevantes sobre la ejecución total o parcial en diferentes segmentos de la ejecución.

Los principales eventos representados en la línea de ejecución son tres:

- Valores de tiempo

- Flags, correspondientes a eventos
- Líneas de comunicación entre procesos

El funcionamiento de Paraver está basado en un filtro que accede directamente al fichero de la traza y facilita al módulo semántico una visión parcial o total de la traza.

El módulo semántico es la clave de la capacidad expresiva de Paraver. Su función es la de generar un valor numérico para cada objeto para ser representado. Para cada objeto, se procesa una función de tiempo que es calculada desde los registros de la traza para representar posteriormente la visualización.

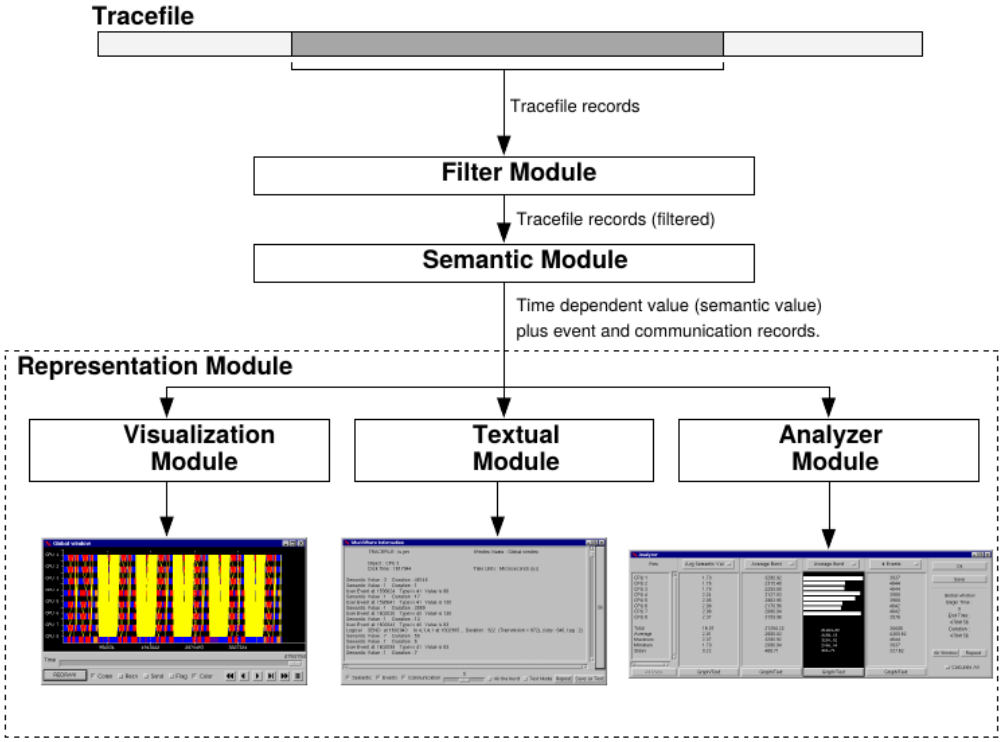


Figura 5.1: Diagrama de procesamiento de trazas en Paraver

5.3 Extrae

Extrae es el software principal destinado a la generación de trazas para Paraver en un análisis post-mortem del programa. Este software utiliza diferentes mecanismos de interposición para inyectar código en la aplicación a analizar y obtener información relacionada con el rendimiento de su ejecución. A continuación se detallan los modelos de programación y los sistema para los que se encuentra disponible.

Plataformas	Modelos de programación
Linux clusters (x86-64)	MPI
BlueGen/Q	OpenMP*
Cray	CUDA*
nVidia GPU	OpenCL*
Intel Xeon Phy	pthreads*
ARM	OmpSs*
Android	Java
K computer	Python
FX10	

Tabla 5.1: Plataformas y modelos de programación soportados por Extrae

Los destacados con un (*) pueden utilizarse en conjunto con MPI.

5.3.1 Ficheros de trazas

Estos ficheros se generan a través de Extrae o Dimemas. En el caso expuesto, se generaran las trazas a través de la instrumentación del programa previamente a su ejecución, obteniendo un fichero o traza que contendrá las diferentes métricas generadas en la ejecución sobre el sistema cluster.

Un fichero de trazas está compuesto con tres tipos de información:

- Estados: asociados al valor de un estado de un thread en un intervalo de tiempo.
- Eventos: representan un evento particular que ha ocurrido en un objeto analizado.
- Relación: como un evento relacionado entre dos objetos en el tiempo.

Esta información se distribuye en el conjunto de tres ficheros:

- Actividad de la aplicación analizada con formato **(.prv)**
- Etiquetas asociadas a los valores numéricos en formato **(.pcf)**
- Utilización de recursos en formato **(.row)**

5.3.2 Mecanismos de interposición

Extrae cuenta con mecanismos de interposición para añadir monitores a la aplicación, no importa que mecanismo se use, el objetivo es el mismo, obtener métricas de rendimiento para proporcionar al analista la relación sobre rendimiento y ejecución. Los mecanismos de interposición disponibles son:

- **Linked Preload (LD_PRELOAD):** La mayoría de sistemas aceptan la inyección de librerías compartidas en una aplicación antes de que esta se cargue en memoria. Si la librería contiene los mismos símbolos que la librería precargada, los símbolos pueden utilizarse para inyectar código en esas llamadas de funciones. En los sistemas Linux este método está disponible a través de la variable LD_PRELOAD. Extrae proporciona soporte para esta técnica inyectando librerías para la mayoría de runtimes integrando funciones que obtienen las métricas adecuadas para su posterior análisis.

- **DynInst:** Esta librería de instrumentación permite la modificación de la aplicación inyectando código en secciones específicas del código fuente del programa.

En el caso expuesto, se utilizará el enlace de la librería instrumentada mediante el mecanismo LD_PRELOAD.

5.3.3 Mecanismos de muestreo

Extrae no solo ofrece la posibilidad de instrumentar el código de la aplicación, también ofrece el uso de mecanismos de muestreo para obtener información acerca de rendimiento. La capacidad de añadir monitores en lugares específicos de la aplicación genera información fácilmente vinculante con el código fuente, que además está relacionada directamente con el flujo de control de la aplicación. La capacidad de añadir este muestreo a la ejecución permite proporcionar información importante de rendimiento en regiones de código específicas.

Existen dos mecanismos de muestreo, el primero basado en los temporizadores de señal, que liberan el controlador de muestreo en un intervalo de tiempo específico, y el segundo, basado en los contadores de rendimiento del procesador. Para obtener estas métricas Extrae utiliza la interfaz PAPI, permitiendo obtener información sobre múltiples componentes del sistema como discos, red, sistema operativo y otros.

Cabe añadir, que algunas versiones de procesadores no son compatibles con el acceso de lectura a registros y contadores.

5.4 Objetivos

La combinación de uso de Extrae y Paraver ofrecen un enorme potencial para el analista, tanto de forma cuantitativa como cualitativa. Con estas herramientas, se facilita el hecho de identificar cuellos de botella y eventos que degradan la ejecución de una aplicación, debido a la visualización del programa desde una vista mucho más detallada, así como comprender mejor el despliegue que realiza la aplicación sobre la arquitectura donde se ejecuta.

El objetivo principal de este capítulo consistirá en la utilización de estas herramientas a modo de **introducción**, generando archivos de trazas con la herramienta **Extrae**, instrumentando el código de los programas utilizados en el sistema cluster a modo de medida de rendimiento para tratar de observar algunos fenómenos con el visualizador **Paraver**.

5.5 Configuración e instalación de tools

A continuación se describirá el proceso para configurar e instalar las herramientas, contando que para la instrumentación de código con Extrae existen algunas dependencias de librerías, las cuales también será necesario compilar e instalar. Es importante tener en cuenta, que sabor de MPI se utiliza en el sistema, ya que la herramienta Extrae está disponible para diferentes variantes. En el sistema cluster desarrollado se utiliza la implementación de OpenMPI. En el caso del visualizador, no es necesario utilizarlo en los mismos ordenadores del sistema cluster, se puede utilizar para el análisis de las trazas un ordenador externo, en nuestro caso con arquitectura x86_64.

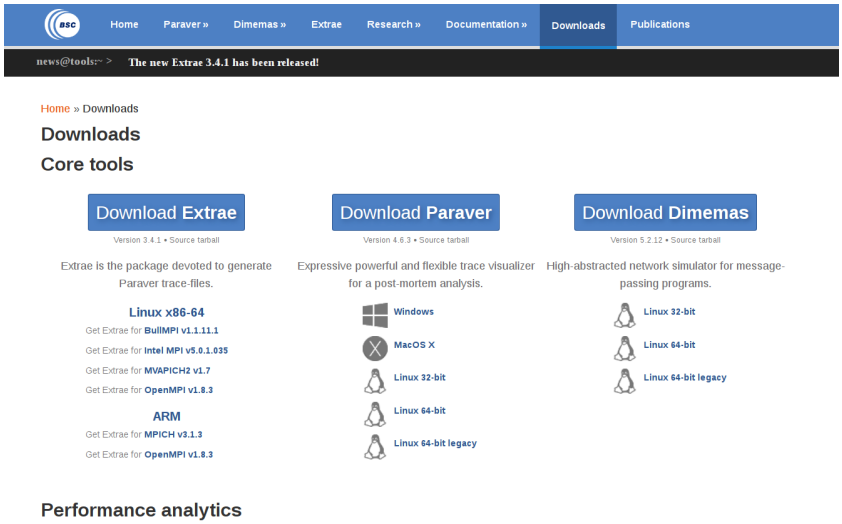


Figura 5.2: Página de descarga oficial de las BSC-Tools

Los source de las herramientas se pueden desde la página oficial del BSC, donde se puede encontrar bastante documentación sobre su funcionamiento y configuración más avanzada. Los links de descarga son los siguientes:

- **Extrae para OpenMPI v1.8.3:**
  `https://ftp.tools.bsc.es/extrae/extrae-latest_linux-x86_64_openmpi+libgomp4.9.tar.bz2`
- **Paraver para Linux 64bits:**
  `https://ftp.tools.bsc.es/paraver/wxparaver-latest-linux-x86_64.tar.bz2`

  El funcionamiento de Extrae depende directamente de las siguientes librerías:
- **PAPI:**
  `http://icl.cs.utk.edu/projects/papi/downloads/papi-5.5.0.tar.gz`
- **libunwind:**
  `http://download.savannah.gnu.org/releases/libunwind/`

Primero se tendría que realizar la compilación e instalación de las dependencias de Extrae, mediante el proceso **./configure**, **make** y **make install**. En caso de que se quiera definir una ruta de instalación manualmente se puede incluir en el primer paso (configure), añadiendo el siguiente parámetro: **-install-prefix=(ruta de instalación deseada)**.

Una vez instaladas las dependencias, se puede proceder a realizar la instalación de Extrae, el cual requiere de una configuración en el comando **./configure** para definir que parámetros y utilidades que se necesitan de la herramienta. Algunos de los requisitos son obligatorios.

```
[zxcv90@cluster]#] ./configure --prefix=/home/zxcv90/extrae/ --with-mpi=/opt/openmpi/--
    without-dyninst --with-unwind=/home/zxcv90/BSCTOOLS/libs/finallibunwind/ --with-
    papi=/home/zxcv90/BSCTOOLS/libs/finalpapi/ --with-papi-headers=/home/zxcv90/
    BSCTOOLS/libs/papi-5.5.0/--enable-instrument-io --enable-posix-clock
```

- **–prefix** Define la ruta de instalación
- **–with-mpi** Define la ruta donde se encuentra MPI

- –**with-unwind** Define la ruta de la librería libunwind
- –**without-dyninst** Sin instrumentación dinámica
- –**with-papi** Ruta de PAPI
- –**with-papi-headers** Ruta de las cabeceras de PAPI
- –**enable-instrument-io** Habilitar instrumentación I/O
- –**enable-posix-clock** Usar rutinas adaptables a cambios de frecuencia de procesador

Si se configura correctamente debería de mostrar un sumario como este con los detalles de la configuración aplicada.

```
Package configuration for Extrae 3.4.1
-----------------------
Installation prefix: /home/zxcv90/extrae
Cross compilation: no
CC: gcc
CXX: g++
Binary type: 32 bits

MPI instrumentation: yes
MPI home: /opt/openmpi/
MPI launcher: /opt/openmpi//bin/mpirun
Fortran decoration: 0 underscores
mixed C/Fortran libraries? no
shared libraries? yes
MPI capabilities: 1-sided I/O
Load-Balancing hooks? no
OpenMP instrumentation: yes, through LD_PRELOAD
GNU OpenMP: yes, libgomp 4.2
IBM OpenMP: no
Intel OpenMP: yes
OMPT: no
OpenSHMEM instrumentation: no
pThread instrumentation: yes
Support for pthread_barrier_wait: yes
CUDA instrumentation: no
OpenCL instrumentation: no
Java instrumentation: unsupported

Performance counters: yes
Performance API: PAPI
PAPI home: /home/zxcv90/BSCTOOLS/libs/finalpapi/
Sampling support: yes

PEBS sampling: no

libbfd available: yes (/usr/lib/)
libiberty available: yes (/usr/lib/)
zlib available: yes (/usr)
libxml2 available: yes (/usr)
BOOST available: no
callstack access: through libunwind (/home/zxcv90/BSCTOOLS/libs/finallibunwind/)

Dynamic instrumentation: no
```

A continuación ya se podría instalar la herramienta Extrae en el directorio deseado con **make** y **make install**.

Cabe destacar que el visualizador Paraver no requiere compilación, ya que la descarga incluye el binario para la arquitectura deseada para su utilización.

5.6 Generación de trazas de ejecución con Extrae

A continuación, se detallará el proceso necesario para instrumentar el código de las aplicaciones a ejecutar con Extrae y poder generar un archivo de trazas válido para su posterior análisis con el programa Paraver.

Para la generación de trazas es necesario definir algunas variables de sistema para que apunten a la ruta donde están los archivos de Extrae.

export EXTRAE_HOME=/export/home/zxcv90/extrae

source $EXTRAE_HOME/etc/extrae.sh

Para poder realizar la interposición de las librerías de instrumentación, se utilizará un script que realizará la precarga de la librería deseada antes de la ejecución. La interposición se realiza en el cargador de runtimes, substituyendo los símbolos originales del código por la librería que se facilite, internamente modificada para poder obtener las métricas de la ejecución.

En el caso expuesto, se quieren instrumentar aplicaciones que utilizan MPI, donde el código fuente puede alternar entre C o Fortran77. Todas las librerías de interposición disponibles se pueden encontrar en el directorio de Extrae bajo la carpeta /lib/. Dependiendo de si se realiza la instrumentación de ejecución para C o Fortran77, se debe de alternar entre las dos opciones disponibles: libmpitrace.so (C) o libmpitracef.so (Fortran)

El script que lanzará la instrumentación deberá de alojarse en el mismo directorio donde este el binario del programa que queramos instrumentar, junto al fichero de configuración extrae.xml.

Estos dos ficheros se pueden encontrar en las siguientes rutas:

- Fichero de configuración: **share/example/MPI/extrae.xml**
- Script de precarga: **share/example/MPI/ld-preload/trace.sh**

El contenido del script es el siguiente. Se pueden diferenciar las dos versiones de export tanto para la instrumentación de aplicaciones desarrolladas con Fortran o aplicaciones desarrolladas en C.

```
#!/bin/bash
source /home/zxcv90/extrae/etc/extrae.sh
export EXTRAE_CONFIG_FILE=extrae.xml
#export LD_PRELOAD=${EXTRAE_HOME}/lib/libmpitrace.so # For C apps
export LD_PRELOAD=${EXTRAE_HOME}/lib/libmpitracef.so # For Fortran apps
## Run the desired program
$*
```

El contenido del fichero de configuración extrae.xml incluye los eventos a monitorizar, así como los modelos de programación para los que se aplicará la instrumentación del código. A continuación se muestran algunos detalles del contenido del fichero.

```
<?xml version='1.0'?>

<trace enabled="yes"
home="/home/zxcv90/extrae"
initial-mode="detail"
type="paraver"
>
```

```
<mpi enabled="yes">
<counters enabled="yes" />
</mpi>

<openmp enabled="yes">
<locks enabled="no" />
<counters enabled="yes" />
</openmp>

<pthread enabled="no">
<locks enabled="no" />
<counters enabled="yes" />
</pthread>

<callers enabled="yes">
<mpi enabled="yes">1-3</mpi>
<sampling enabled="no">1-5</sampling>
<dynamic-memory enabled="no">1-3</dynamic-memory>
</callers>

<user-functions enabled="no" list="/home/bsc41/bsc41273/user-functions.dat" exclude-
    automatic-functions="no">
<counters enabled="yes" />
</user-functions>

<counters enabled="yes">
<cpu enabled="yes" starting-set-distribution="1">
<set enabled="yes" domain="all" changeat-time="0">
PAPI_TOT_INS,PAPI_TOT_CYC,PAPI_L1_DCM,PAPI_L2_DCM,PAPI_L3_TCM,PAPI_BR_INS,PAPI_BR_MSP,
    RESOURCE_STALLS
</set>
<set enabled="yes" domain="all" changeat-time="0">
PAPI_TOT_INS,PAPI_TOT_CYC,PAPI_VEC_SP,PAPI_SR_INS,PAPI_LD_INS,PAPI_FP_INS
<sampling enabled="no" period="1000000000">PAPI_TOT_CYC</sampling>
</set>
</cpu>
<network enabled="no" />
<resource-usage enabled="no" />
<memory-usage enabled="no" />
</counters>

<storage enabled="no">
```

Para la instrumentación de una aplicación, en este caso utilizando MPI, basta con interponer el script de instrumentación antes de la ejecución del programa previamente compilado. Manteniendo una sintaxis como la siguiente:

```
[zxcv90@torrex]# mpirun -np 2 -machinefile 2mach ./trace.sh[Script de precarga] ./[
    Aplicacion MPI]
```

Si la ejecución y la instrumentación finaliza correctamente, esto generará los ficheros con las trazas en el formato adecuado para su posterior análisis con la aplicación Paraver, en formato .prv. Cuando se disponga de estos archivos ya se puede realizar un análisis del comportamiento de la aplicación y visualizar los diferentes eventos y estados que se han desarrollado.

```
[zxcv90@torrex trazasNP2]# ls -l
[...] libmpi_f77.so.1.0.3.pcf
[...] libmpi_f77.so.1.0.3.prv
[...] libmpi_f77.so.1.0.3.row
[...] set-0
[...] TRACE.mpits
[...] TRACE.sym
```

Es importante almacenar e identificar las trazas generadas con la configuración de ejecución del programa, diferenciando por ejemplo numero de threads, computadores involucrados, tamaño del problema, etc.

5.7 Análisis Paraver

5.7.1 Introducción al GUI

Para poder visualizar los archivos de trazas generados y realizar los análisis post-ejecución, es necesario la ejecución de la aplicación Paraver, clasificada dentro de la suite BSC Tools como la herramienta destinada a la visualización de los ficheros generados previamente. Para utilizar la herramienta, basta con descomprimir el fichero descargado y ejecutar el binario contenido: **wxparaver.bin**.

La ejecución del binario mostrará la interfaz gráfica de la aplicación. El aspecto será similar al siguiente:

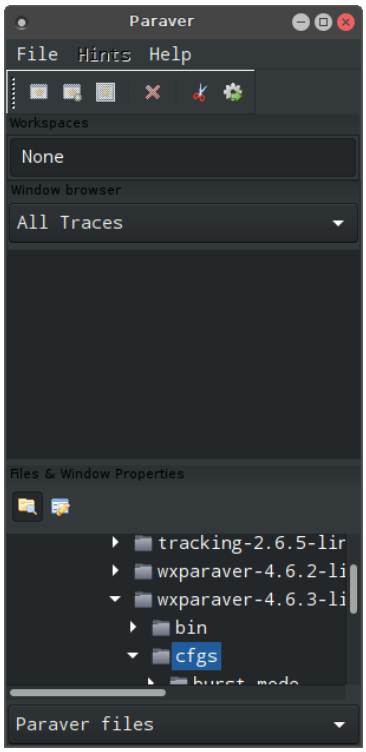


Figura 5.3: Ventana principal de Paraver

Desde el menú **File >Load Trace** se puede cargar un fichero de traza generado, donde la extensión adecuada es .prv.

Cuando el fichero de trazas está cargado, se puede generar una visualización previa de la traza en general pulsando sobre el siguiente botón **New Single Timeline Window**.

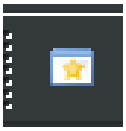


Figura 5.4: Botón de generación de Timeline principal de una traza

Esta acción generará una ventana nueva con una visualización de la aplicación, en este caso, del Kernel EP ejecutado en dos nodos. En el **eje Y** se representan los diferentes hilos de ejecución, desplegados, independientemente de que estén distribuidos en diferentes máquinas o no, en el **eje X** se representa el tiempo total de la ejecución del programa o en su defecto un tiempo parcial de la ejecución si se ha recortado la sección previamente.

Las **líneas amarillas** en la visualización por defecto, señalizan las líneas de comunicación que se han generado durante la ejecución del programa, indicando que hilos han establecido comunicación directa entre si mediante paso de mensajes.

Los colores representados, muestran diferentes estados o valores en los que se encuentra la variable u objeto representado en cada fracción de la ejecución, ajustándose adecuadamente a la duración temporal. De esta forma, se desvelan diferentes tipos de comportamiento que ha podido adoptar la ejecución del programa en cada punto en concreto que se haya registrado en la generación de las trazas.

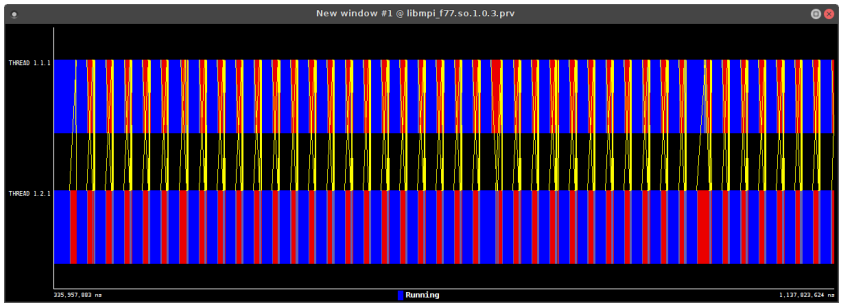


Figura 5.5: Detalle de visualización de traza en Paraver

Paraver ofrece la posibilidad de mostrar la misma información de diferente manera, para ello, desde el menú desplegable con el botón derecho del raton sobre la ventana de Timeline, es posible aplicar diferentes configuraciones de vista o informaciones, tal como podrían ser, flags de eventos, diferentes representaciones gráficas, tablas de valores, etc.

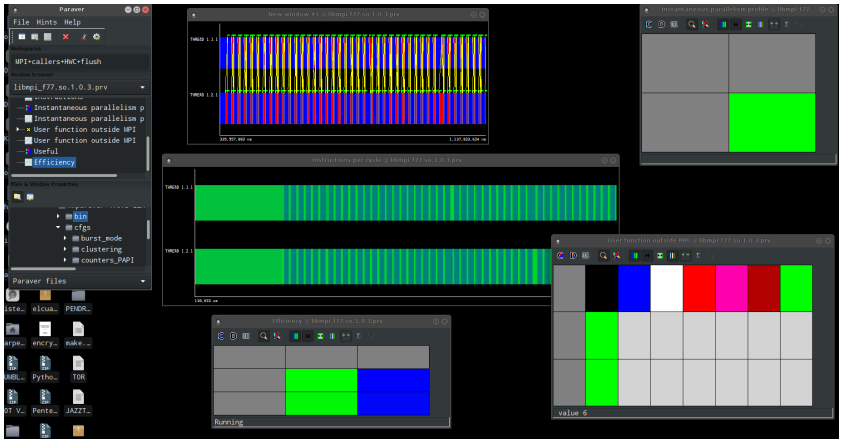


Figura 5.6: Diferentes visualizaciones con la aplicación Paraver

Por ejemplo, en la visualización generada previamente, considerándola como la ventana principal representando el timeline de la traza. La codificación cromática de colores corresponde a las diferentes acciones que han adoptado los threads en ejecución adecuándose a la leyenda que representa la siguiente imagen.

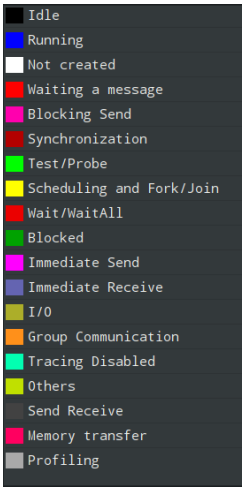


Figura 5.7: Leyenda de código de colores para estados de thread

Uno de los puntos fuertes de esta herramienta, es la opción de aplicar visualizaciones variadas que correspondan a diferentes métricas obtenidas durante la instrumentación del programa. Estas configuraciones/vistas se dividen dentro de estos grandes subgrupos, relacionadas con modelos de programación concretos, métricas de hardware específicas de la arquitectura o medidas de recursos generales del sistema.

Grupos configuraciones aplicables
burst_mode
clustering
counters_PAPI
CUDA
folding
General
Java
mpi
OmpSs
OpenCL
OpenMP
otf
pthread
sampling+folding
scripts
software_counters
spectral

Tabla 5.2: Lista de los distintos subgrupos de visualizaciones disponibles

Dentro de cada grupo, se pueden encontrar más subgrupos o directamente los ficheros de configuración que una vez aplicados generan la vista correspondiente a la métrica, evento o visualización que se necesite ver.

En el caso expuesto, no se pretende ahondar en el análisis en precisión de una traza con la utilización de Paraver, pero se pretende realizar una aproximación sencilla a su utilización general para presentarla como una aplicación extremadamente útil en la generación de perfiles para el análisis de rendimiento y eficiencia en aplicaciones paralelas.

5.7.2 Metodología

En el siguiente apartado se procederá al proceso de análisis de los diferentes ficheros de trazas generadas durante la ejecución de algunos de los programas utilizados anteriormente en las medidas de rendimiento NAS Parallel Benchmarks.

Se pretende aproximar el comportamiento general de las aplicaciones desplegadas sobre el sistema multicomputador implementado y ver algunos de los fenómenos y efectos principales que se manifiestan en la ejecución de una aplicación con un modelo de programación orientado al paralelismo.

Para el análisis con Paraver se utilizarán las trazas generadas en los kernels siguientes:

Kernel EP: Kernel que en paralelización con 4 nodos ofrece un rango de mejora con un speedup máximo de 4.03 puntos.

Kernel CG: En clase B, Kernel que más tiempo demora en su ejecución en paralelo.

Sobre estos ficheros de trazas generados, se aplicarán distintas visualizaciones para observar y obtener la información que pueda contener cada archivo de trazas, con el objetivo de visuali-

zar si se revela alguna información importante que puede esclarecer el funcionamiento de las aplicaciones ejecutándose sobre el sistema real.

Algunos de los puntos que se pretenden observar, son los siguientes:

- **Estados de comunicación MPI**
- **Grado de paralelismo, patrones**
- **Estado de los threads desplegados**

5.7.3 Análisis de trazas: Kernel EP - Class C

Para efectuar el análisis en la interfaz gráfica de Paraver, se procedería a la carga del fichero de trazas previamente generado para el programa, tal como se ha detallado anteriormente. Posteriormente, se aplicaría la previsualización principal descrita.

Figura 5.8: Timeline principal de la traza del Kernel EP con 4 threads

Si se analiza la visualización, se muestran un total de 4 hilos desplegados sobre el eje Y, donde los colores de las barras horizontales representan el estado de cada thread.

Como se puede visualizar, de los 4 hilos desplegados, la mayor parte del tiempo permanecen en estado un estado azul, que en este caso corresponde al estado de Running (ejecutándose), por lo que a primera vista se podría deducir que durante la ejecución del programa en general la mayor parte del tiempo los hilos permanecen en ejecución por el dominio del color azul y que solo existe sincronización entre los threads en las fases de principio y final, que muestran colores diferentes.

Si se analizan los códigos de colores facilitados en la leyenda del visor. Se pueden diferenciar los diferentes estados que adoptan los hilos desplegados en cada instante de tiempo durante la ejecución.

A continuación se mostrarán distintas visualizaciones para realizar un análisis introductorio de un archivo de trazas.

- **Histograma MPI: cfgs/mpi/analisys/mpi_Stats.cfg**

  Esta configuración muestra inicialmente un histograma con el que se podría generar una tabla con el % de tiempo que cada thread de manera individual consume en cada llamada MPI.

  La visualización predeterminada del histograma es la siguiente:

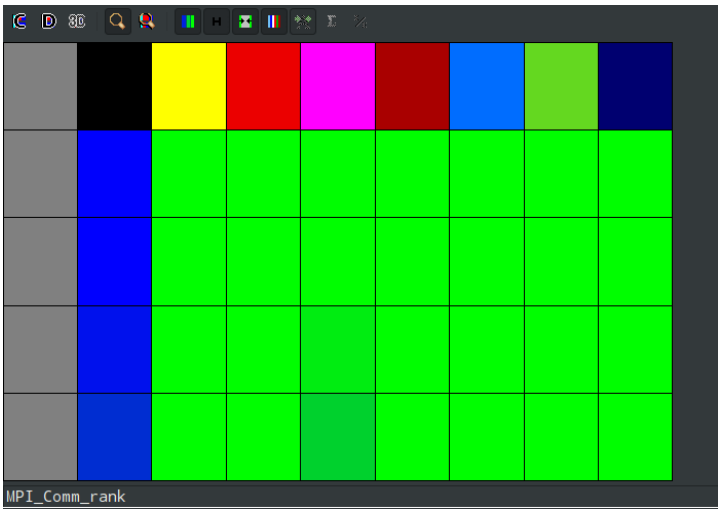


Figura 5.9: Histograma de llamadas en MPI

En el histograma se representan los hilos y los diferentes estados con una relación cromática directamente a los valores y porcentajes de las diferentes llamadas MPI utilizadas.

Resulta más interesante desde un histograma visualizar una tabla de estadísticas y datos numéricos aplicando el botón de la primera lupa.

Figura 5.10: Botón generación estadísticas

Esta acción genera una tabla de datos numéricos relacionados directamente con el ámbito que se este analizando en ese momento. A partir de los datos generados, se puede interpretar de manera general que llamadas MPI predominan en tiempo de ejecución.

Por ejemplo, el campo **Average** representa la eficiencia paralela de la aplicación, la entrada **Avg/Max** representa el balance de carga global que existe en la aplicación, pudiendo valorar si la paralelización se realiza de manera equilibrada entre los distintos recursos hardware disponibles, mientras que la entrada **Maximum** representa la eficiencia en la comunicación. Según la documentación oficial, si los valores representados son inferiores al 85 % se recomienda analizar diferentes métricas con más detalle para determinar un empobrecimiento en el rendimiento final de la aplicación desde el punto de vista de llamadas MPI.

	Outside MPI	MPI_Bcast	MPI_Barrier	MPI_Allreduce	MPI_Comm_rank	MPI_Comm_size	MPI_Init	MPI_Finalize
THREAD 1.1.1	99.99 %	0.00 %	0.00 %	0.00 %	0.01 %	0.00 %	0.00 %	0.00 %
THREAD 1.2.1	99.58 %	0.00 %	0.00 %	0.40 %	0.00 %	0.00 %	0.00 %	0.00 %
THREAD 1.3.1	87.80 %	0.00 %	0.00 %	12.19 %	0.00 %	0.00 %	0.00 %	0.00 %
THREAD 1.4.1	82.72 %	0.00 %	0.00 %	17.27 %	0.01 %	0.00 %	0.00 %	0.00 %
Total	370.09 %	0.00 %	0.01 %	29.86 %	0.02 %	0.00 %	0.00 %	0.01 %
Average	92.52 %	0.00 %	0.00 %	7.46 %	0.01 %	0.00 %	0.00 %	0.00 %
Maximum	99.99 %	0.00 %	0.00 %	17.27 %	0.01 %	0.00 %	0.00 %	0.00 %
Minimum	82.72 %	0.00 %	0.00 %	0.00 %	0.00 %	0.00 %	0.00 %	0.00 %
StDev	7.48 %	0.00 %	0.00 %	7.48 %	0.00 %	0.00 %	0.00 %	0.00 %
Avg/Max	0.93	0.42	0.67	0.43	0.62	0.83	0.73	0.79

Figura 5.11: Sumario generado desde histograma con valores de llamadas MPI

- **Llamadas MPI: cfgs/mpi/views/MPI_CALL.cfg**

  La carga de la siguiente configuración, genera una visualización que corresponde a las fracciones de tiempo donde se encuentran las llamadas realizadas a través de una anotación MPI, bien para tareas de sincronización o transferencias directas de datos.

  La representación de la configuración ofrece una vista como la siguiente:

Figura 5.12: Visualización de llamadas MPI ubicadas sobre timeline

La visualización desvela que las llamadas MPI durante la ejecución de la aplicación se realizan mayoritariamente al final de la ejecución.

La interfaz Paraver, para permitir un análisis más preciso y con más detalle, permite realizar zoom sobre zonas concretas del timeline representado, esto se puede conseguir seleccionando la zona deseada para efectuar la ampliación, con el propio mouse. Al realizar zoom sobre las zonas del principio y el final, se obtienen las visualizaciones siguientes.

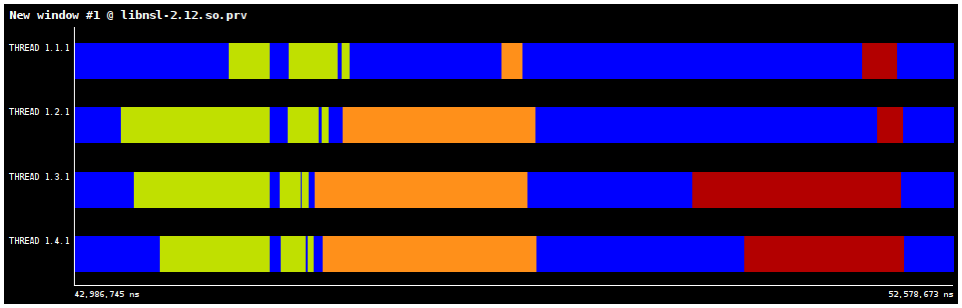


Figura 5.13: Zoom sobre el timeline donde se observan diferentes estados de thread al inicio de la ejecución

Entre el tiempo de ejecución **42,987 us** y **52,579 us** con una duración de **9,592 us** se realiza la sincronización inicial de los threads con el hilo principal tal como se puede observar. La visualización muestra de forma ampliada sobre el timeline con los threads representados, los diferentes estados por los que transita cada hilo, pasando del estado inicial azul (Running) a estados de sincronización como Group Communication en naranja o Synchronization en rojo.

En la imagen siguiente se muestra el código de colores con la información relativa a las llamadas realizadas en MPI tal como se muestra en la configuración cargada previamente.

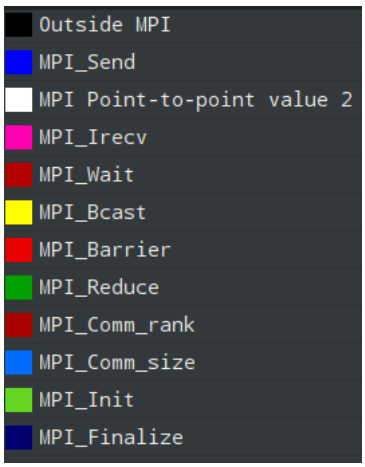


Figura 5.14: Código de colores correspondientes de las llamadas en MPI

Figura 5.15: Diferentes tipos de llamadas en MPI coloreadas

- **Paralelismo instantáneo: cfgs/General/views/Instantaneous_parallelism.cfg**

  La siguiente configuración, muestra de manera lineal el patrón del grado paralelismo que se alcanza en la aplicación ejecutada. La línea dibujada sobre el eje temporal, representa el incremento del grado de paralelismo desplegado durante la ejecución, indicando en que zonas predomina la ejecución paralela del programa.

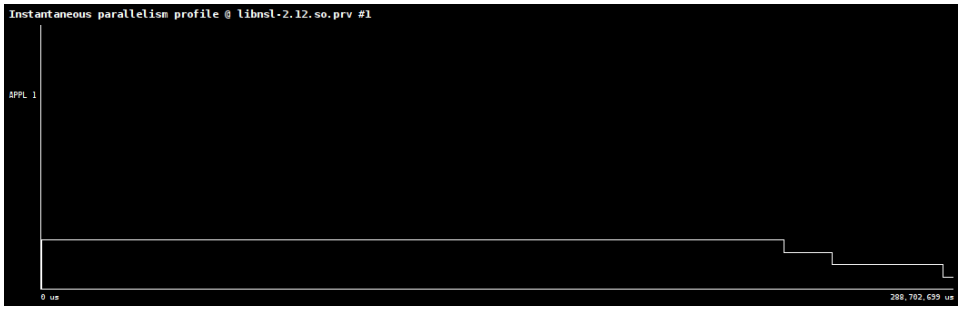


Figura 5.16: Función lineal constante relativa al paralelismo de la aplicación

Tal como se puede observar en la imagen anterior, el grado de paralelismo mantiene una línea estable mientras los hilos de ejecución desempeñan el cálculo parcial del resultado del kernel y representa al final de la ejecución el decremento de paralelismo cuando los threads entran en fase de sincronización para reducir los resultados parciales a uno solo mediante sumarización.

- **Eficiencia de ejecución: config/general/analisys/efficiency.cfg**

  Paraver ofrece la opción de generar histogramas con graduación de colores, o la obtención de tablas con datos numéricos para analizar distintas métricas, tal como se ha explicado anteriormente. Una de estas configuraciones se trata del histograma relacionado con la eficiencia de la ejecución del programa.

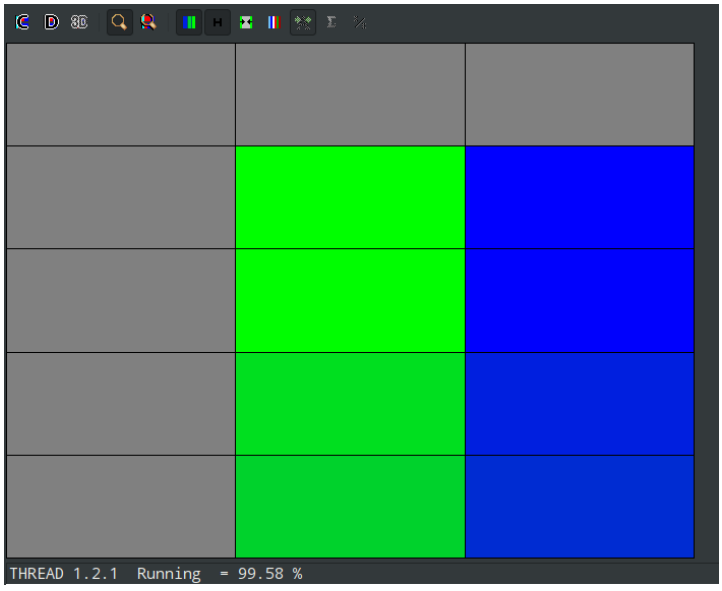


Figura 5.17: Histograma de utilización de los diferentes threads

Si se desplaza el ratón sobre las diferentes celdas de la tabla en la barra de estado inferior se muestra el estado de cada thread y que porcentaje ha tenido en cada estado durante la ejecución. Además, el botón de la lupa superior genera una tabla de datos numéricos que se pueden guardar en formato (.csv).

	Idle	Running
THREAD 1.1.1	0.01 %	99.99 %
THREAD 1.2.1	0.42 %	99.58 %
THREAD 1.3.1	12.20 %	87.80 %
THREAD 1.4.1	17.28 %	82.72 %
Total	29.91 %	370.09 %
Average	7.48 %	92.52 %
Maximum	17.28 %	99.99 %
Minimum	0.01 %	82.72 %
StDev	7.48 %	7.48 %
Avg/Max	0.43	0.93

Figura 5.18: Valores de tasas de ejecución o espera de los threads

Analizando los valores de la tabla obtenida, los threads ejecutados en el Kernel se encuentran en funcionamiento durante todo el tiempo de ejecución del programa, con una media del 92.52 %. Este valor indica un alto aprovechamiento del recurso de procesadores para la resolución de la aplicación, por lo que puede estimar que la eficiencia de la ejecución es óptima, pudiendo observar un ligero desbalanceo entre threads.

5.7.4 Análisis de trazas: Kernel CG - Class B

En el siguiente apartado se realizará un análisis de la traza generada en la ejecución del Kernel CG - Class B. **Este Kernel es el que demora más tiempo en su ejecución paralela**, el análisis superficial de la traza tratará de desvelar algunos aspectos clave en el rendimiento del programa.

Primero se debería de proceder a la carga de la traza tal como se ha explicado anteriormente desde la interfaz de Paraver. La previsualización de la aplicación es la siguiente:

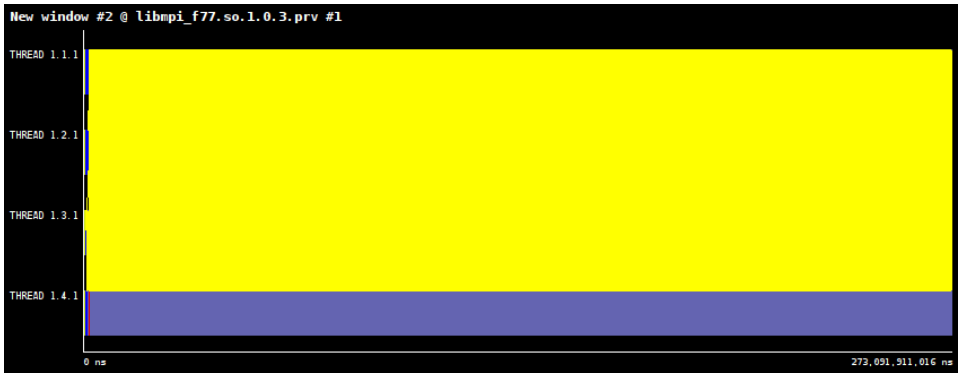


Figura 5.19: Visualización timeline de la traza para el Kernel CG con líneas de comunicación en amarillo

Como se puede observar, predomina el color amarillo, que corresponde exactamente a múltiples líneas de comunicación entre los 4 hilos desplegados durante la ejecución de la aplicación.

Desde el botón derecho sobre la ventana de Timeline, se puede deshabilitar la visualización de las líneas de comunicación existentes, quedando la vista como en la siguiente figura.

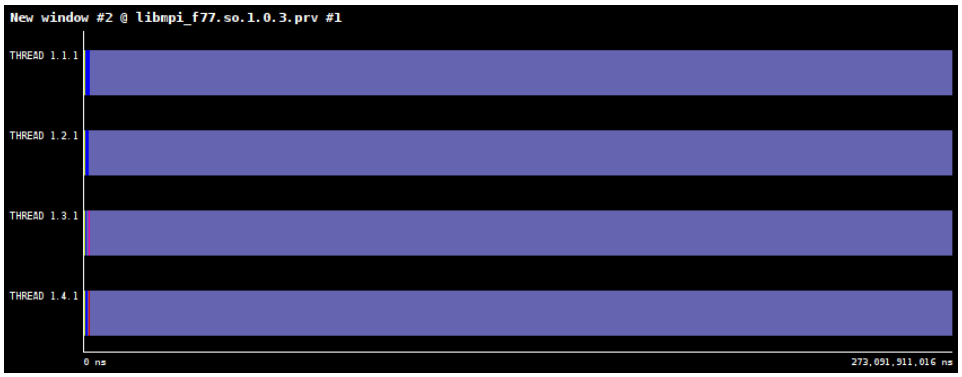


Figura 5.20: Visualización timeline de la traza para el Kernel CG sin líneas de comunicación

Los colores que se visualizan corresponden con el siguiente código de colores.

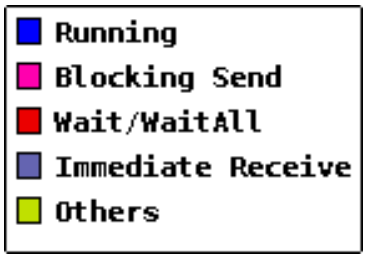


Figura 5.21: Código de colores relativo a estados de threads

Aparentemente, en la ejecución predomina un estado de Inmediate Receive, por lo que se podría intuir que predominan estados de espera de recepcion de datos en los 4 hilos desplegados. Si se aplica zoom sobre una zona aleatoria del Timeline, el resultado desvela el siguiente detalle.

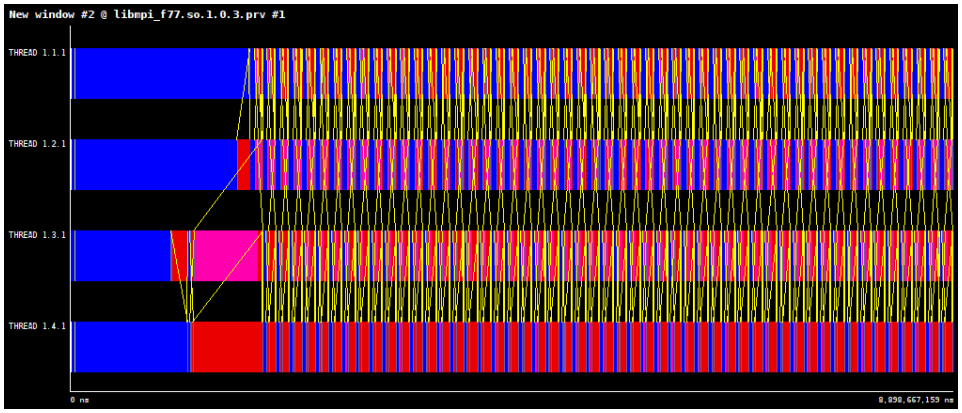


Figura 5.22: Ampliación 1 de la vista de la traza para apreciar los estados de threads y la alta comunicación entre hilos

Existe una alta comunicación entre los hilos desplegados debido al intercambio constante de datos para la sincronización entre los threads. Si se aplica un aumento del zoom para una visualización con mayor precisión, se puede observar como predominan estados de espera (Wait), donde el tiempo de ejecución de los hilos está absorbido por tiempos dedicados a la espera de la recepción de datos para poder proseguir con la secuencia del programa.

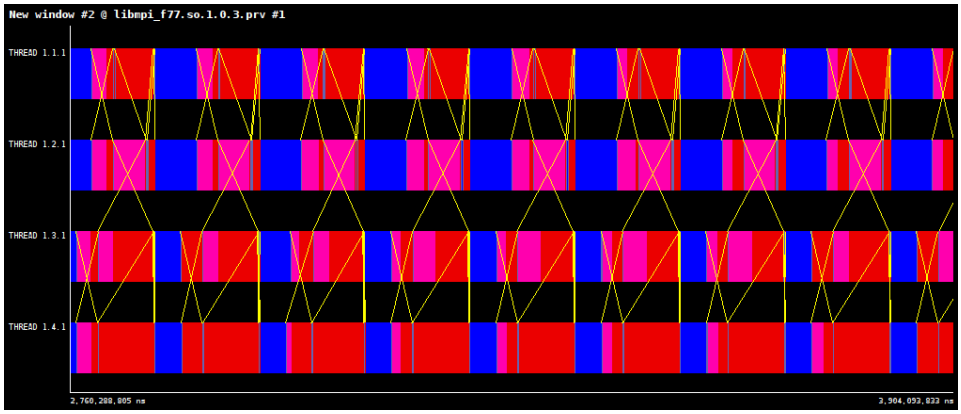


Figura 5.23: Ampliación 2 de la vista de la traza para apreciar los estados de threads y la alta comunicación entre hilos

Comparativa: 4 vs 2 threads

Si se comparan las visualizaciones anteriores con la ejecución del mismo programa pero desplegado sobre únicamente dos threads de ejecución, se pueden apreciar las siguientes diferencias: al inicio del programa no se encuentran secciones de espera tan largas, además de que la sincronización entre hilos se realiza de forma más directa, reduciendo el tiempo residual destinado a sincronización entre hilos.

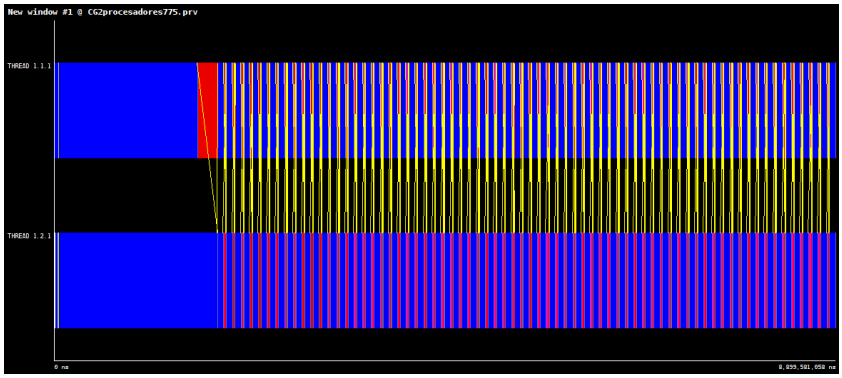


Figura 5.24: Vista principal de la traza obtenida de la ejecución en 2 hilos

En la segunda captura aplicando un aumento de zoom sobre una sección de la representación de tiempo, se puede apreciar como los tiempos de espera entre los threads son mucho más reducidos, demorando menos tiempo dedicado a este estado, directamente, se despliegan menos líneas de comunicación entre ellos, por lo que el aprovechamiento del recurso de procesador es más optimizado.

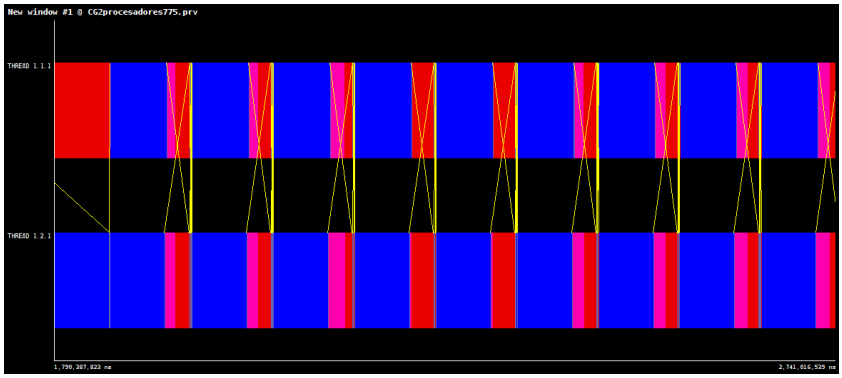


Figura 5.25: Ampliación de la vista de la traza de ejecución en dos hilos con mejor tiempo dedicado a estados de espera

- **Llamadas MPI: cfgs/mpi/analisys/mpi_Stats.cfg**

  Esta configuración muestra un histograma con el que se puede generar una tabla con el % de tiempo que cada thread de manera individual consume en cada llamada MPI. Si se analizan las estadísticas se puede interpretar de manera general que llamadas MPI predominan durante la ejecución. El campo Average usualmente representa la eficiencia paralela de la aplicación, la entrada Avg/Max representa el balance de carga global de la aplicación, mientras que la entrada Maximum representa la eficiencia en la comunicación. Si los valores son inferiores al 85 % se recomienda analizar diferentes métricas con más detalle que pueden influir directamente sobre el rendimiento de la aplicación.

	Outside MPI	MPI_Send	MPI_Irecv	MPI_Wait	MPI_Bcast	MPI_Barrier	MPI_Reduce	MPI_Comm_rank	MPI_Comm_size	MPI_Init	MPI_Finalize
THREAD 1.1.1	43.28 %	12.58 %	0.47 %	43.67 %	0.00 %	0.00 %	0.00 %	0.00 %	0.00 %	0.00 %	0.00 %
THREAD 1.2.1	42.90 %	42.84 %	0.56 %	13.70 %	0.00 %	0.00 %	0.00 %	0.00 %	0.00 %	0.00 %	0.00 %
THREAD 1.3.1	26.84 %	34.63 %	0.19 %	38.34 %	0.00 %	0.00 %	0.00 %	0.00 %	0.00 %	0.00 %	0.00 %
THREAD 1.4.1	27.22 %	12.90 %	0.22 %	59.66 %	0.00 %	0.00 %	0.00 %	0.00 %	0.00 %	0.00 %	0.00 %
Total	140.24 %	102.96 %	1.43 %	155.36 %	0.00 %	0.00 %	0.00 %	0.00 %	0.00 %	0.00 %	0.01 %
Average	35.06 %	25.74 %	0.36 %	38.84 %	0.00 %	0.00 %	0.00 %	0.00 %	0.00 %	0.00 %	0.00 %
Maximum	43.28 %	42.84 %	0.56 %	59.66 %	0.00 %	0.00 %	0.00 %	0.00 %	0.00 %	0.00 %	0.00 %
Minimum	26.84 %	12.58 %	0.19 %	13.70 %	0.00 %	0.00 %	0.00 %	0.00 %	0.00 %	0.00 %	0.00 %
StDev	8.03 %	13.32 %	0.16 %	16.50 %	0.00 %	0.00 %	0.00 %	0.00 %	0.00 %	0.00 %	0.00 %
Avg/Max	0.81	0.60	0.64	0.65	0.73	0.78	0.54	0.85	0.89	0.73	0.96

Figura 5.26: Tabla de valores de histograma de tasa de utilización de llamadas MPI en ejecución de 4 hilos Kernel CG - B

Comparativa: 4 vs 2 threads

Analizando la traza generada con 2 hilos de ejecución se pueden destacar algunos porcentajes de tiempo en algunos estados de llamadas MPI. Los estados de espera en la ejecución de 4 hilos representa el 38.84 % del tiempo de ejecución, mientras que en la ejecución de 2 hilos representa solo el 9.72 % de espera. En ejecución de instrucciones útiles del programa fuera de llamadas MPI, podemos observar un 35.06 % de tiempo de ejecución en el programa de 4 hilos frente un 71 % de tiempo de ejecución en el de 2 hilos, con lo cual, se podría determinar que la ejecución con 2 hilos en este programa es más eficiente.

	Outside MPI	MPI_Send	MPI_Irecv	MPI_Wait	MPI_Bcast	MPI_Barrier	MPI_Reduce	MPI_Comm_rank	MPI_Comm_size	MPI_Init	MPI_Finalize
THREAD 1.1.1	72.12 %	17.48 %	0.26 %	10.14 %	0.00 %	0.00 %	0.00 %	0.00 %	0.00 %	0.00 %	0.00 %
THREAD 1.2.1	69.88 %	20.52 %	0.29 %	9.30 %	0.00 %	0.00 %	0.00 %	0.00 %	0.00 %	0.00 %	0.00 %
Total	142.00 %	38.00 %	0.55 %	19.44 %	0.00 %	0.00 %	0.00 %	0.00 %	0.00 %	0.00 %	0.00 %
Average	71.00 %	19.00 %	0.28 %	9.72 %	0.00 %	0.00 %	0.00 %	0.00 %	0.00 %	0.00 %	0.00 %
Maximum	72.12 %	20.52 %	0.29 %	10.14 %	0.00 %	0.00 %	0.00 %	0.00 %	0.00 %	0.00 %	0.00 %
Minimum	69.88 %	17.48 %	0.26 %	9.30 %	0.00 %	0.00 %	0.00 %	0.00 %	0.00 %	0.00 %	0.00 %
StDev	1.12 %	1.52 %	0.02 %	0.42 %	0.00 %	0.00 %	0.00 %	0.00 %	0.00 %	0.00 %	0.00 %
Avg/Max	0.98	0.93	0.94	0.96	0.61	0.92	0.91	0.95	0.93	0.59	0.94

Figura 5.27: Tabla de valores de histograma de tasa de utilización de llamadas MPI en ejecución de 2 hilos Kernel CG - B

- **Llamadas MPI: cfgs/mpi/views/MPI_CALL.cfg**

  La siguiente configuración, muestra sobre la línea temporal los momentos exactos donde se producen llamadas MPI, de esta manera, se ofrece la posibilidad de observar los momentos exactos donde se producen las llamadas que realizan los diferentes hilos desplegados, y el tipo de llamada que se efectúa.

  Los colores de las los distintos eventos representados corresponden directamente con el siguiente código de colores.

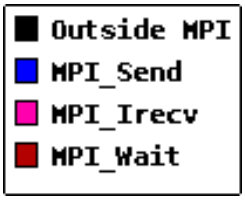


Figura 5.28: Código de colores de llamadas MPI

  La configuración muestra generalmente una visualización como la mostrada en la imagen siguiente, donde en este caso, se puede observar una constante ejecución de llamadas MPI durante el tiempo de ejecución total de la aplicación.

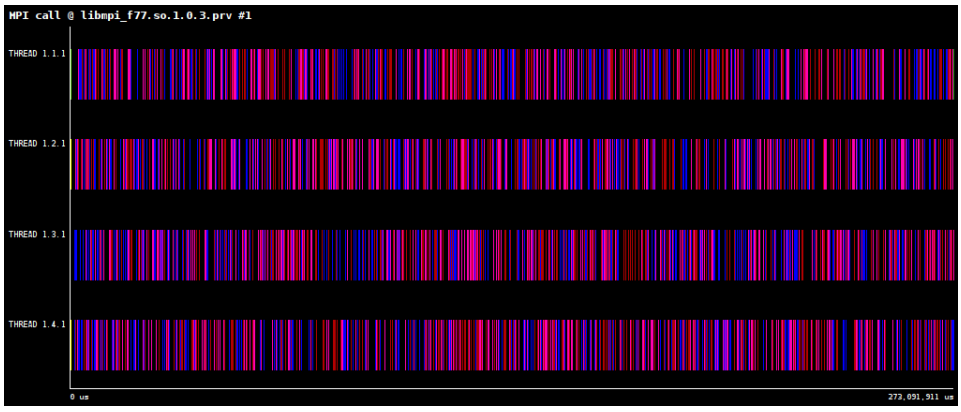


Figura 5.29: Visualización de timeline con las llamadas MPI realizadas durante la ejecución

- **Paralelismo instantáneo: cfgs/General/views/Instantaneous_parallelism.cfg**

  Tal como se detallaba anteriormente, la visualización del paralelismo instantáneo permite intuir el patrón de despliegue paralelo, representado sobre la línea de tiempo donde se concentra una ejecución con mayor nivel de paralelismo, a nivel general de la aplicación.

  La visualización siguiente muestra el patron de paralelismo que sigue la aplicación ejecutada, mostrando el incremento y decremento reiterado entre las secciones de carga computacional y donde la comunicación se representa con las líneas amarillas dibujadas en la zona superior, coincidiendo exactamente entre los diferentes patrones que sigue la aplicación.

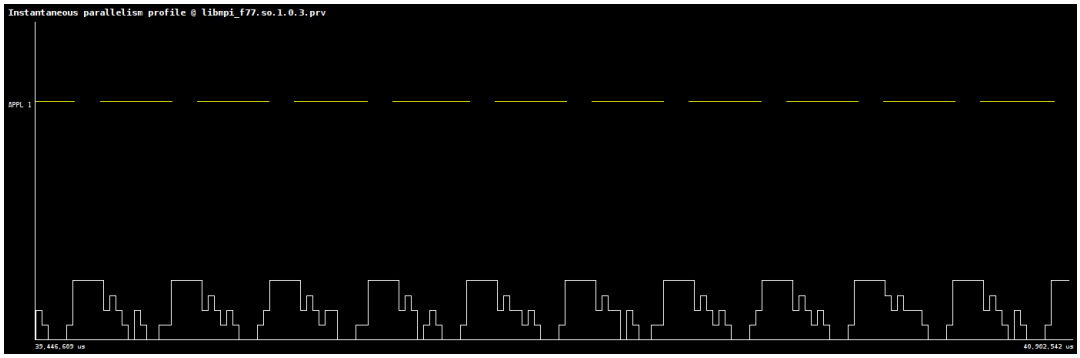


Figura 5.30: Visualización lineal de patrón de paralelismo en la aplicación

- **Eficiencia de ejecución: config/general/analisys/efficiency.cfg**

  La configuración de eficiencia, visualiza el histograma que muestra el porcentaje de tiempo destinado al tiempo útil de ejecución en cada uno de los hilos desplegados en la ejecución tal como se ha explicado anteriormente.

  Si se visualiza la tabla de valores numéricos, se facilita el análisis estadístico de las diferentes métricas representadas.

	Idle	Running
THREAD 1.1.1	56.74 %	43.26 %
THREAD 1.2.1	57.11 %	42.89 %
THREAD 1.3.1	73.18 %	26.82 %
THREAD 1.4.1	72.79 %	27.21 %
Total	259.82 %	140.18 %
Average	64.96 %	35.04 %
Maximum	73.18 %	43.26 %
Minimum	56.74 %	26.82 %
StDev	8.03 %	8.03 %
Avg/Max	0.89	0.81

Figura 5.31: Tabla de tasas de rendimiento de 4 threads desplegados

  Los porcentajes en los que cada hilo de forma independiente se mantiene en espera superan todos el 50 %, con una media total del 64,96 % de tiempo de ejecución en estados de espera,

indicando un claro desaprovechamiento de recursos de procesador. Esto puede inducir a que se intuya que la alta sincronía entre hilos y la eficiencia de esta aplicación dependa en gran medida de un recurso muy importante en el sistema cluster, la red de comunicación.

Comparativa: 4 vs 2 threads

En la visualización del grado de eficiencia final del programa en la ejecución con dos hilos, se permite apreciar la diferencia de rendimiento entre los dos tipos de ejecución lanzada.

El 64,96 % del tiempo en la ejecución de 4 hilos está destinado por estados de esperas, destinando solo el 34,04 % a tiempo de ejecución útil.

En cambio, en la ejecución con 2 hilos se encuentran valores del 29,02 % de estados de espera y un 70,98 % del total de tiempo en estado de ejecución, mostrando claramente una mayor eficiencia en esta última configuración.

	Idle	Running
THREAD 1.1.1	27.90 %	72.10 %
THREAD 1.2.1	30.13 %	69.87 %
Total	58.03 %	141.97 %
Average	29.02 %	70.98 %
Maximum	30.13 %	72.10 %
Minimum	27.90 %	69.87 %
StDev	1.12 %	1.12 %
Avg/Max	0.96	0.98

Figura 5.32: Tabla de tasas de rendimiento de 2 threads desplegados

- **Ancho de banda Nodos (MB/s): config/mpi/views/node_bandwith.cfg**

  La siguiente configuración aplicada, permite una visualización que representa el ancho de banda en MB/s de la comunicación entre nodos a través de la red de interconexión del cluster.

  Debido al alto nivel de configuración de Paraver, existen diferentes representaciones para visualizar los valores obtenidos de la medida del análisis de la métrica. La visualización de una gradiente de colores dependiendo del valor, permite interpretar el ancho de banda disponible en relación con la siguiente escala de colores.

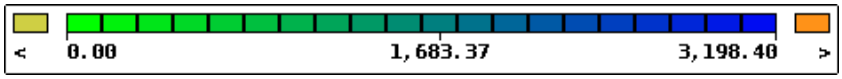


Figura 5.33: Escala cromática de valores en el ancho de banda entre nodos

A continuación se visualizan los valores obtenidos durante la ejecución del programa en 4 procesos individuales distribuidos entre las 4 máquinas del cluster. Como se puede apreciar, el ancho de banda entre los diferentes computadores difiere, obteniendo mejores valores en los nodos 1 y 4 observando una descompensación inusual.

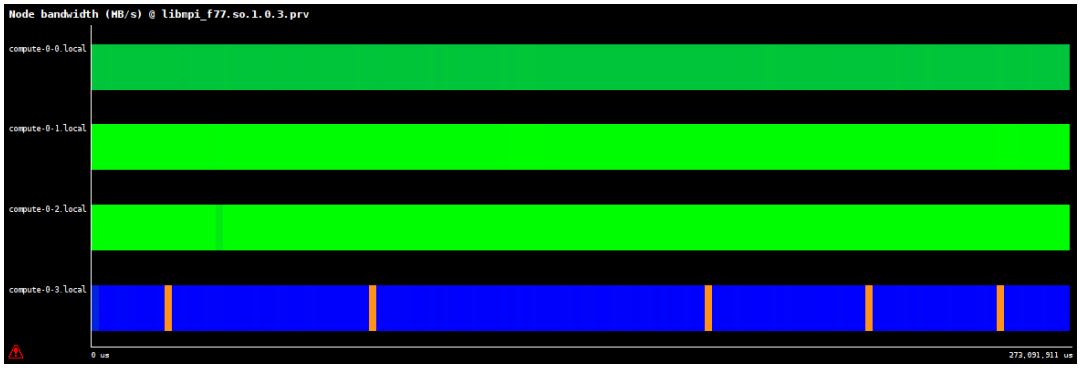


Figura 5.34: Visualización timeline del ancho de banda en 4 nodos

Comparativa: 4 vs 2 threads

Si se compara la misma visualización con la ejecución entre dos procesos distribuidos, se obtiene el siguiente patrón de comunicación.

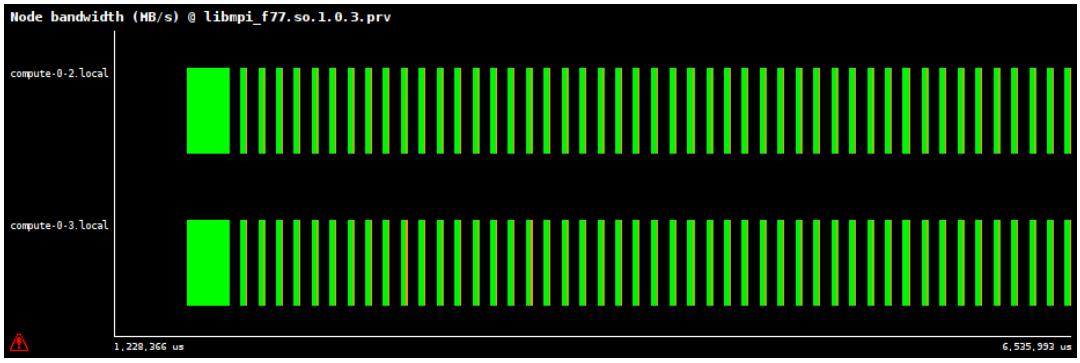


Figura 5.35: Visualización timeline del ancho de banda en 2 nodos

El patrón de comunicación resulta más intermitente y regular, donde la saturación de la red de interconexión es menor, debido a un menor intercambio de información a través de la red, este hecho aplica una menor exigencia de ancho de banda por parte de los nodos, liberando la sobrecarga del recurso de red, y permitiendo una ejecución mucho más ágil destinando más tiempo útil de procesador al cálculo de la solución del kernel.

- **Ancho de banda Procesos (MB/s): config/mpi/views/proc_bandwith.cfg**

  La siguiente configuración, muestra una visualización que representa el ancho de banda en MB/s de la comunicación entre procesos. La vista está representada en una gradiente de colores en relación con la siguiente escala de colores, similar a la comentada anteriormente.

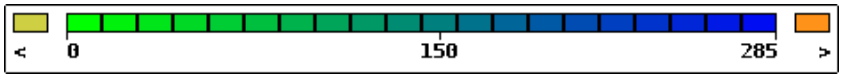


Figura 5.36: Escala de valores y colores del ancho de banda entre procesos

Tal como se puede observar, disponen del máximo ancho de banda la transferencia de datos entre los threads 1 y 4, obteniendo menor valor de ancho de banda los threads

2 y 3, mostrando un alto porcentaje de tiempo destinado a recursos de comunicación desbalanceado entre los hilos.

Figura 5.37: Visualización timeline del ancho de banda en procesos

Comparativa: 4 vs 2 threads

Si se compara la visualización anterior contra la ejecución en 2 hilos, se puede observar que en esta última, los registros de comunicación están mucho más segmentados, obteniendo dentro de la escala de colores unos valores más altos de ancho de banda, a la vez que más balanceados entre los dos procesos, con una menor congestión del recurso de red.

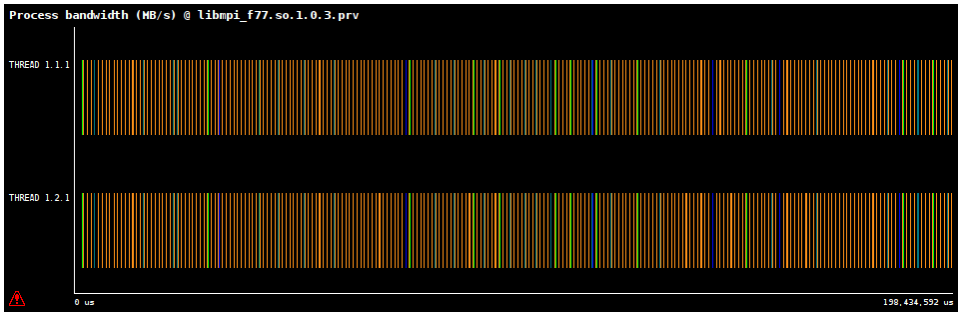


Figura 5.38: Visualización timeline del ancho de banda en 2 nodos

5.8 Conclusiones

La utilización de este tipo de herramientas en entornos de supercomputación pueden ayudar a los investigadores a visualizar de manera global el comportamiento de los distintos programas ejecutados dentro de sus sistemas, aproximando el análisis post-ejecución para poder entender o corregir posibles factores que degraden directamente el rendimiento y que interfieran en alcanzar el máximo punto de eficiencia en la ejecución.

Las herramientas Extrae y Paraver, desarrolladas en el Barcelona Supercomputing Center, ofrecen un software extremadamente potente y altamente configurable para obtener múltiples métricas de la ejecución de programas en entornos donde el rendimiento sea la clave.

En el caso expuesto, la utilización de estas herramientas es meramente introductoria, para dar a conocer de manera sencilla las ventajas que pueden llegar a ofrecer. Aún así, se ha realizado un

análisis superficial que desvela el comportamiento real de las aplicaciones cuando se despliegan sobre el sistema hardware real, permitiendo observar algunos fenómenos característicos que ayudan a entender el funcionamiento del modelo de programación por paso de mensajes y pueden ayudar a desvelar los motivos de una ineficiencia dada en la ejecución de programas sobre la infraestructura.

Con la información obtenida, en un proceso de experimentación o caso real, se podrían basar mejoras para corregir los distintos problemas encontrados y verificados con las herramientas, como pudieran ser la degradación del rendimiento debido a la saturación de la interconexión entre máquinas, balanceo incorrecto de la carga de computo, etc. Una vez identificadas las posibles causas que degradan el rendimiento, se puede enfocar de forma más directa la correción necesaria para evitar la perdida de rendimiento en la ejecucion, pudiendo implicar medidas a nivel de código del programa o medidas a nivel de infraestructura, etc.

En los ejemplos mostrados anteriormente, el proceso se ha basado simplemente en la comparación de ejecuciones distintas, para observar las diferencias existentes entre diferentes configuraciónes. En caso de estar desarrollando una aplicación basada en modelos de programación paralelos para ejecucion sobre la infraestructura, se deberian de definir las causas exactas de la degradacion de rendimiento y aplicar una solucion posible. Sobre el sistema implementado en este proyecto, se podria tratar de mejorar la red de interconexion, pero al contar con hardware limitado y equipamiento antiguo este paso no se puede realizar.

Capítulo 6

Consumo eléctrico

Uno de los factores que más connotación tienen en entornos que integran este tipo de infraestructuras y sistemas, es el factor del consumo eléctrico. Generalmente, los supercomputadores o clusters están conformados por un gran número de equipos pudiendo variar entre decenas y miles de ellos. Este equipamiento, de manera propia puede tener un consumo eléctrico asumible si se habla de unas pocas unidades, pero el hecho de integrar un gran número dispara el consumo de electricidad y más si la carga de trabajo es alta durante un periodo largo de tiempo.

6.1 Objetivos

En este capítulo, se tratará de calcular y aproximar el consumo eléctrico del sistema cluster implementado durante su funcionamiento, para observar las medidas obtenidas bajo diferentes cargas de trabajo y el coste económico que puede acarrear el funcionamiento de la máquina en un contrato de electricidad doméstico.

6.2 Útil de mesura

Para realizar la medida de los valores en el cluster, se utilizará un medidor de consumo eléctrico doméstico.

Este útil dispone de un enchufe integrado donde calcula la potencia eléctrica utilizada de todos los dispositivos que estén conectados directamente y los muestra en un display pudiendo mostrar diferentes métricas relacionadas con el consumo eléctrico.

El útil se trata del siguiente modelo: **Floureon TS-836A**

Características

- Medidas diversos parámetros: Potencia (W), energía (kWh), voltios, amperios, hertzios, factor de potencia y la potencia máxima (W)
- Acumulativo Monitor de kilovatio-hora
- Gastos Calculados de electricidad
- Muestra Voltios, amperios y potencia exacta a 3 %, bajo consumo de energía.

Especificaciones

- Precisión: 0.5W
- Pantalla Frecuencia: 50HZ
- Intensidad máxima: Max 16A
- Rango de voltaje: 230V-250V
- Enchufe: 2-pin de EU

Figura 6.1: Herramienta de medición de consumo eléctrico utilizada: Floureon TS-836A

6.3 Metodología

Para proceder a la medida de consumo eléctrico se seguiran las siguientes pautas:

- Medida individual nodo en estado idle
- Medida individual nodo en estado 100 % carga de trabajo
- Medida cluster en estado idle
- Medida cluster en estado 100 % carga de trabajo

El hardware implicado en las distintas mediciones incluirá:

- Nodo Frontend
- Nodo cálculo 1 (compute-0-0)
- Nodo cálculo 2 (compute-0-1)
- Nodo cálculo 3 (compute-0-2)

- Nodo cálculo 4 (compute-0-3)
- Switch de interconexión

Una vez se obtengan las diferentes medidas descritas anteriormente se procederá a estimar el coste aproximado del sistema en diferentes rangos de tiempo considerando que se encuentra al 100 % de su funcionamiento. Este coste se estimará basándose en las tarifas aplicadas a una vivienda por un distribuidor de electricidad nacional.

6.4 Medidas de consumo eléctrico

En los siguientes apartados se detallarán las diferentes medidas de consumo eléctrico realizadas en los diferentes estados.

6.4.1 Medida de nodos individuales en estado idle

Este apartado contiene las mediciones realizadas sobre los diferentes nodos del sistema de forma individualizada mientras se encuentran en un estado de espera con la mínima carga computacional.

Sistema	Estado	W	KWh
Frontend node	Idle	61,5	0,615
compute-0-0	Idle	48,5	0,485
compute-0-1	Idle	48,5	0,485
compute-0-2	Idle	92,4	0,924
compute-0-3	Idle	81,5	0,815

Tabla 6.1: Medidas de consumo eléctrico en los diferentes nodos de manera individual en un estado de espera

6.4.2 Medida de nodos individuales en estado 100 carga de trabajo

Este apartado contiene las mediciones realizadas sobre los diferentes nodos del sistema de forma individualizada mientras se encuentran en un estado de trabajo con un 100 % de carga computacional.

Sistema	Estado	W	KWh
Frontend node	Run	73,5	0,0735
compute-0-0	Run	87,5	0,0875
compute-0-1	Run	87,5	0,0875
compute-0-2	Run	134	0,134
compute-0-3	Run	142,5	0,1425

Tabla 6.2: Medidas de consumo eléctrico en los diferentes nodos de manera individual en un estado de funcionamiento pleno

6.4.3 Medida de cluster en estado idle

Este apartado contiene las mediciones realizadas sobre el sistema cluster al completo (1 nodo gobierno + 4 nodos de cálculo) mientras se encuentra en un estado de reposo sin una alta carga computacional.

Sistema	Estado	W	KWh
cluster (all)	Idle	320,5	0,3

Tabla 6.3: Medidas de consumo eléctrico el sistema cluster en un estado de espera

6.4.4 Medida de cluster en estado 100 % carga de trabajo

Este apartado contiene las mediciones realizadas sobre el sistema cluster al completo (1 nodo gobierno + 4 nodos de cálculo) mientras se encuentra bajo una carga de trabajo computacional al 100 % aproximadamente.

Sistema	Estado	W	KWh
cluster (all)	Run	500,5	0,5005

Tabla 6.4: Medidas de consumo eléctrico el sistema cluster en un estado de funcionamiento pleno

6.5 Coste eléctrico de funcionamiento

El siguiente apartado del capítulo se basará en el cálculo del coste económico que provocaría la utilización del sistema cluster en utilización a pleno rendimiento, utilizando una línea de consumo de una vivienda estándar con las tarifas aplicadas por un proveedor de electricidad nacional.

6.5.1 Precio actual electricidad

Basándose en la facturación doméstica de la vivienda donde está instalado el sistema cluster implementado, la facturación del consumo se desglosaría de la siguiente manera:

Energía		
Potencia facturada	3,3 KW x 34 días x 0,123955€/KW día	13,91€
Energía	**(ener. consumida)** x 0,155895€/KWh	X
Coste total sin impuestos		13,31€+ X
Impuesto electricidad	5,11269632 %/Coste total sin imp.	X
Total energía		X€
IVA 21 %		X€
Total importe		X€

Tabla 6.5: Medidas de consumo eléctrico el sistema cluster en un estado de funcionamiento pleno

En las próximas secciones, esta tabla se adecuará con la medida de consumo eléctrico (Kwh) obtenida con el sistema cluster funcionando en torno a un 100 % de carga de trabajo durante diferentes periodos de tiempo en un día integro, (1 hora, 2 horas, 6 horas, 12 horas, 24 horas (día completo)).

6.5.2 Estimación de coste consumo eléctrico sistema cluster

A continuación, en los diferentes apartados se estimará el coste de electricidad aproximado del sistema cluster en los diferentes rangos de tiempo definidos anteriormente, donde se expondrá el consumo total aproximado en el rango temporal así como el impacto económico que tendría de manera mensual en la facturación del proveedor de electricidad.

Estimación de coste 1 hora utilización al día

Coste económico basado en 1 hora de utilización al día del sistema cluster durante un ciclo de facturación (1 mes).

Potencia consumida (kWh)= 500,5
Potencia consumida mes(kWh/mes)= 15,015

Energía		
Potencia facturada	3,3 KW x 34 días x 0,123955€/KW día	13,91€
Energía	**15,02** x 0,155895€/KWh	2,34€
Coste total sin impuestos		16,25€
Impuesto electricidad	5,11269632 %/Coste total sin imp.	0,8307215485€
Total energía		17,08€
IVA 21 %		3,59€
Total importe		20,67€

Tabla 6.6: Medidas de consumo eléctrico el sistema cluster en un estado de funcionamiento pleno

Coste consumo eléctrico 1 mes= 20,67€

Estimación de coste 2 horas utilización al día

Coste económico basado en 2 horas de utilización al día del sistema cluster durante un ciclo de facturación (1 mes).

Potencia consumida (kWh)= 1,001
Potencia consumida mes(kWh/mes)= 30,03

Energía		
Potencia facturada	3,3 KW x 34 días x 0,123955€/KW día	13,91€
Energía	**30,03** x 0,155895€/KWh	4,68€
Coste total sin impuestos		18,59€
Impuesto electricidad	5,11269632 %/Coste total sin imp.	0,9503954194€
Total energía		19,54€
IVA 21 %		4,10€
Total importe		23,64€

Tabla 6.7: Medidas de consumo eléctrico el sistema cluster en un estado de funcionamiento pleno

Coste consumo eléctrico 1 mes= 23,64€

Estimación de coste 6 horas utilización al día

Coste económico basado en 6 horas de utilización al día del sistema cluster durante un ciclo de facturación (1 mes).

Potencia consumida (kWh)= 3,003
Potencia consumida mes(kWh/mes)= 90,09

Energía		
Potencia facturada	3,3 KW x 34 días x 0,123955€/KW día	13,91€
Energía	**90,09** x 0,155895€/KWh	14,04€
Coste total sin impuestos		27,95€
Impuesto electricidad	5,11269632 %/Coste total sin imp.	1,429090903€
Total energía		29,38€
IVA 21 %		6,17€
Total importe		35,55€

Tabla 6.8: Medidas de consumo eléctrico el sistema cluster en un estado de funcionamiento pleno

Coste consumo eléctrico 1 mes= 35,55€

Estimación de coste 12 horas utilización al día

Coste económico basado en 12 horas de utilización al día del sistema cluster durante un ciclo de facturación (1 mes).

Potencia consumida (kWh)= 6,006
Potencia consumida mes(kWh/mes)= 180,18

Energía		
Potencia facturada	3,3 KW x 34 días x 0,123955€/KW día	13,91€
Energía	**180,18** x 0,155895€/KWh	28,09€
Coste total sin impuestos		42,00€
Impuesto electricidad	5,11269632 %/Coste total sin imp.	2,15€
Total energía		44,14€
IVA 21 %		9,27€
Total importe		53,41€

Tabla 6.9: Medidas de consumo eléctrico el sistema cluster en un estado de funcionamiento pleno

Coste consumo electrico 1 mes= 53,41€

Estimación de coste 24 horas utilización/día completo

Coste económico basado en 24 horas de utilización al día del sistema cluster durante un ciclo de facturación (1 mes).

Potencia consumida (kWh)= 12,012
Potencia consumida mes(kWh/mes)= 360,36

Energía		
Potencia facturada	3,3 KW x 34 días x 0,123955€/KW día	13,91€
Energía	**360,36** x 0,155895€/KWh	56,18€
Coste total sin impuestos		70,09€
Impuesto electricidad	5,11269632 %/Coste total sin imp.	3,583220578€
Total energía		73,67€
IVA 21 %		15,47€
Total importe		89,14€

Tabla 6.10: Medidas de consumo eléctrico el sistema cluster en un estado de funcionamiento pleno

Coste consumo eléctrico 1 mes= 89,14€

6.6 Sumario de consumos

A continuación se detalla una tabla con los distintos costes que generaría la utilización del sistema cluster implementado en diferentes tasas de utilización con una previsión de 2, 6 y 12 meses de uso.

Horas uso día	Pot. (kWh)	Pot. mes (kWh/mes)	1 mes	2 meses	6 meses	12 meses
1 hora día	0,5005	15,015	20,67€	41,33€	124,00€	247,99€
2 horas día	1,001	30,03	23,64€	47,29€	141,86€	283,72€
6 horas día	3,003	90,09	35,55€	71,10€	213,31€	426,62€
12 horas día	6,006	180,18	53,41€	106,83€	320,49€	640,97€
24 horas (día full)	12,012	360,36	89,14€	178,28€	534,84€	1.069,68€

Tabla 6.11: Estimación de costes en función de la utilización del sistema cluster

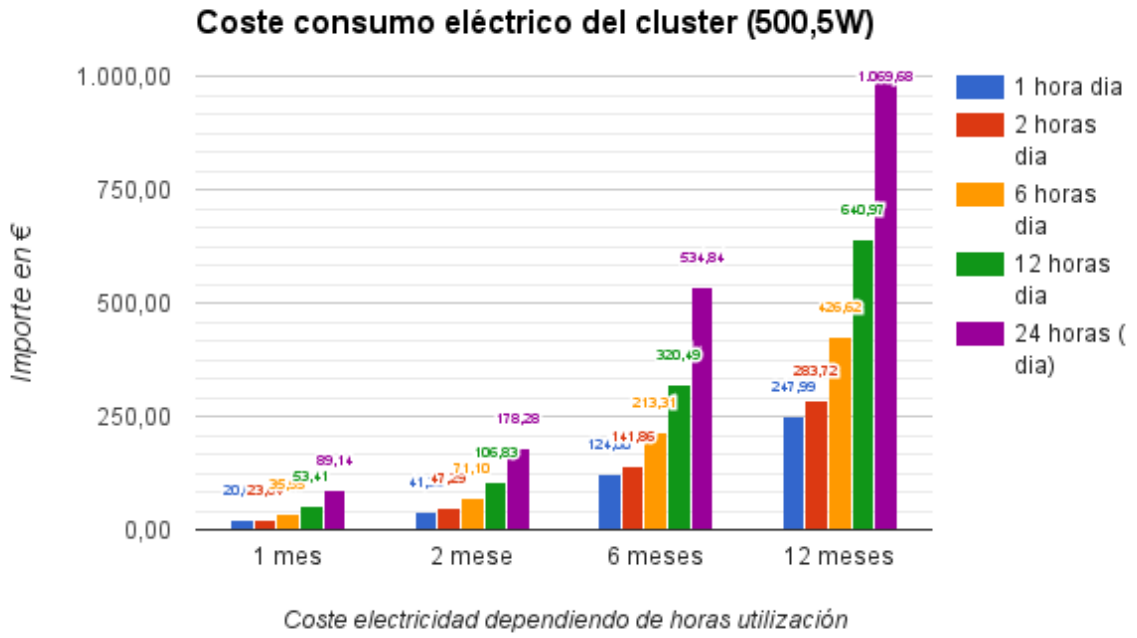


Figura 6.2: Sumario de costes por consumo eléctrico en diferentes rangos de tiempo

6.7 Eficiencia energética

En la actualidad, el desarrollo de nuevas tecnologías en la fabricación de microprocesadores ofrece un aumento de rendimiento mediante la investigación de nuevas arquitecturas, donde un punto que tiene una gran importancia es la eficiencia energética que puedan ofrecer estas nuevas tecnologías.

La combinación de múltiples elementos de cálculo provoca normalmente un aumento de rendimiento, ligado directamente a un aumento de consumo. Para estimar la eficiencia energética de los sistemas se utiliza una magnitud comúnmente conocida como rendimiento por watt (FLOPS/WATT). Esta medida trata de aproximar la eficiencia de una arquitectura de computadores o un hardware básandose en el ratio de capacidad de cálculo entre la energía consumida. Para determinar esta medida se utilizará la siguiente formula.

$$Eficiencia_{energtica} = \frac{Potencia_{clculo}}{Potencia_{electrica}}$$

La potencia de cálculo vendrá determinada por el máximo valor de MFLOPS registrado en la ejecución de los distintos benchmarks ejecutados de la suite.

La potencia eléctrica vendrá determinada por el máximo valor de registro de consumo eléctrico medido en el conjunto del sistema cuando está funcionando a pleno rendimiento.

Los valores obtenidos del sistema son los siguientes:

Potencia cálculo	649.16 MFLOPS
Potencia eléctrica (W)	500,5 W

Tabla 6.12: Valores obtenidos del sistema cluster

A partir de estas medidas se puede obtener el valor de la eficiencia energética.

$$Eficiencia_{energtica} = \frac{649{,}16_{MFLOPS}}{500,5_{W}} = 1,297_{MFLOPS/W}$$

Este valor indicaría el ratio de eficiencia energética que ofrece el sistema desarrollado. En este caso, se trata de un valor muy bajo comparado con los actuales sistemas que incorporan los últimos avances en tecnologías de procesadores. En sistemas de altas prestaciones de supercomputación, la web Top500 ofrece actualmente una clasificación bajo la nomenclatura Green500 donde se clasifican los sistemas más eficientes del mundo, pudiendo verse valores de hasta 9462,1 MFLOPS/W en el sistema actual más eficiente, equipado con la última tecnología NVIDIA DGX-1.

Capítulo 7

Conclusiones

La realización del proyecto se pudo llevar a cabo por la disposición de múltiples equipos en desuso obtenidos de contenedores de reciclaje del campus UPV en Alcoi, lo cual motivó la idea del aprovechamiento de ellos para la configuración de una arquitectura multicomputador.

La configuración del sistema atraviesa inicialmente fases de adecuación del hardware disponible para establecer unas características similares entre los equipos, utilizando hardware compatible o la compra por internet del material necesario, en este caso se adquirió memoria RAM compatible. El coste económico del desarrollo de este proyecto es de carácter asequible, siempre y cuando se disponga de un mínimo de equipamiento que esté retirado o se tenga intención de reciclar, el cual no es difícil de conseguir, por lo que el alcance del desarrollo de este sistema está al alcance de cualquier persona u organización.

El estudio y la investigación para proceder a la instalación de este tipo de arquitectura es necesario, ya que se tiene que analizar qué uso se le va a dar al cluster, debido a que existen diferentes tipos de clusters. A grades rasgos existen clusters orientados a servicios, alta disponibilidad y balanceo de accesos, o los orientados a computación, comúnmente utilizados para investigación e ingeniería para obtener una mayor potencia de cálculo por la suma de equipamiento.

Enfocando el despliegue de la plataforma como un recurso que pudiera servir a la docencia en asignaturas como Sistemas Operativos, Computación Paralela, Arquitectura de computadores, Arquitecturas avanzadas o Lenguajes y Entornos de Programación Paralela, el desarrollo de este tipo de sistema ofrece una plataforma de coste reducido que puede resultar de gran utilidad para enseñar de manera práctica conceptos relacionados en materias englobadas dentro de ambitos como: configuración de sistemas operativos, programación paralela, arquitecturas avanzadas, etc.

La decisión del uso de un sistema OpenSource basado en GNU/Linux, orientado a utilización en clusters como es el kit Rocks, basado en CentOS, ofrece un sistema que simplifica el proceso de instalación y configuración, permitiendo realizar de manera cómoda el despliegue de la infraestructura y permitiendo centrar mayor atención sobre la explotación de la arquitectura en tareas como programación y ejecución. Añadir, que las múltiples y avanzadas configuraciones que permiten realizarse sobre el sistema basado en Linux, pueden generar funcionamientos no deseados tanto en el hardware como en el software, normalmente derivados del desconocimiento de las configuraciones que se aplican sobre el sistema operativo. En este caso, se recomienda la revisión de la documentación oficial de Rocks para conocer los distintos procedimientos de configuración y como deben aplicarse sobre el sistema.

Este tipo de kits de sistema, en este caso orientados a clusters, ofrecen por defecto una amplia gama de software de libre uso orientado a diferentes tareas comúnmente realizadas sobre la infraestructura. Aplicaciones como Ganglia, permiten monitorizar en tiempo real la utilización de recursos hardware en cada máquina instalada en el cluster simplificando en gran medida la revisión de funcionamientos extraños o empobrecimiento del rendimiento. También, en el caso de que se implemente este tipo de sistema en un espacio de una universidad o institución, el sistema ofrece bastantes medidas de seguridad, basándose en la gestión según los estándares de UNIX de permisos y usuarios, cortafuegos basados en IPTABLES y un completo sistema de monitorización del sistema de ficheros descrito en la sección Tripwire.

Con este tipo de infraestructura tipo cluster basándose en los principios de supercomputación, la explotación del hardware resulta atractiva en la ejecución de programas paralelos basados en el modelo de paso de mensajes, disponiendo de varias implementaciones del estándar MPI para su uso. En el caso expuesto, la medida de rendimiento basada en la ejecución de algoritmos pertenecientes a la suite *NASBenchmarks* arroja en la mayoría de los casos resultados positivos en cuanto al incremento de rendimiento por la adición de elementos de cálculo. Esta situación puede considerarse constante en el caso ideal, pero en la realidad esta premisa no siempre se cumple, pudiendo observar rendimientos más óptimos en la ejecución en menor número de nodos ya que en el caso real, otros elementos como la interconexión afectan directamente al rendimiento final del sistema.

Basándose en los valores obtenidos en diferentes métricas, el incremento de rendimiento o speedup dependiendo de la configuración para la ejecución de benchmarks, oscila entre valores aproximados de entre 0,5 y 4 puntos en el mejor caso. Esto determina que en general, la paralelización de los algoritmos explotados, aportan el incremento de rendimiento necesario para mejorar el tiempo de ejecución cuando aparecen problemas clásicos de limitación en ejecución secuencial. Añadir, que basándose en las medidas de los resultados de los benchmarks, se ha obtenido un valor tope de 649,16 MFLOPS, este valor dista mucho de los valores obtenidos actualmente en instalaciones basadas en este principio utilizando tecnologías más actuales o mayor número de nodos, pero sirve para ilustrar el incremento de potencia que se consigue con la inclusión de mayor número de unidades de cálculo de forma relativamente sencilla.

Aprovechando la disposición de este tipo de infraestructura, se puede utilizar para la experimentación y análisis. En el caso desarrollado, la modificación de configuraciones buscando la optimización del código fuente del programa, mediante los *FLAGS* disponibles en los compiladores, sirven para mostrar que determinados cambios permiten ofrecer aportaciones positivas al rendimiento final, reduciendo significativamente el tiempo de ejecución de un programa paralelo en este caso.

También, el análisis post-ejecución con las herramientas Paraver y Extrae, con toda la fase de instalación, configuración y uso que conllevan, además del gran potencial que ofrecen para el analista en un entorno de computación paralela, pueden servir para mostrar que el sistema desplegado puede utilizarse como plataforma de pruebas para investigar nuevas herramientas, y en el caso desarrollado, obtener métricas y resultados interesantes y complejos que pueden ilustrar a efectos prácticos algunos fenómenos tratados en distintas materias. En los resultados obtenidos, se podría considerar por ejemplo la degradación de rendimiento por saturación de la red, la distribución desbalanceada de la carga de trabajo asignada a cada procesador o el desbordamiento de memoria RAM observado.

Sobre las medidas de consumo eléctrico obtenidas, contando solo con 5 máquinas a pleno funcionamiento, se consume aproximadamente unos 0,5 kW/h, que dependiendo del tiempo de uso

del sistema producen un impacto diferente en el coste eléctrico, aunque prácticamente asumible si se desea disponer de una plataforma para experimentación o análisis de pruebas.

Cierto es que la eficiencia eléctrica calculada en el ratio de FLOPS/WATTIOS es bastante ajustada, pudiendo considerar la instalación íntegra como poco eficiente con un valor de 1,3 MFLOPS/W. Se debería de tener en cuenta que se trata de hardware obsoleto, con procesadores fabricado en 90 nm, producidos aproximadamente sobre 2005 o anteriormente, y que los programas ejecutados no estén adecuadamente adaptados a la arquitectura, punto a tener muy en cuenta cuando se busca la explotación de este tipo de sistemas.

En general, el proyecto desarrollado puede resultar interesante porque abarca diferentes campos de la informática, donde la orientación del mismo se ha dirigido hacia finalidades de investigación o uso en docencia. Se podría haber profundizado más en algunos apartados como en la programación eficiente de algoritmos paralelos, pero se ha preferido mostrar el proceso de análisis y creación de la infraestructura y mostrar la capacidad de explotación que ofrece como sistema de cálculo paralelo.

Durante el desarrollo, la aportación de conocimiento es bastante notable, permitiendo experimentar realmente sobre una versión reducida de un sistema multicomputador. De entre los diferentes puntos que abarca entre varios ámbitos de la informática, se podrían destacar algunos como: análisis e implantación del sistema cluster basado en sistema operativo GNU/Linux, administración de configuraciones del sistema GNU/Linux, uso de herramientas software de seguridad, control y monitorización como Ganglia o Tripwire, instalación de software especifico, explotación del hardware mediante modelos de programación de paso de mensajes, comparativas y análisis de resultados y estimación de rendimiento, pruebas de optimización con compiladores y análisis de mejora, análisis post-ejecución con herramientas libres de instituciones de referencia mundial, análisis de costes, etc.

En vistas al futuro, sería interesante desarrollar este tipo de arquitectura con hardware más actualizado y contando con más potencia o disponer de mayor número de elementos para tratar de aproximarlo mucho más a una instalación realista. De poder realizarse esta implementación, el máximo interés sería desplegarlo en una institución pública como el campus de Alcoi para disponer de una plataforma que pudiera ser explotada tanto por el personal docente como por el alumnado interesado en estas temáticas impartidas por algunas asignaturas.

Capítulo 8

Anexos

8.1 Rocks CLI All options

```
add appliance {appliance} [graph=string] [membership=string] [node=string] [os=string]
    [public=bool]
add appliance attr {appliance} {attr} {value} [attr=string] [value=string]
add appliance route {os} {address} {gateway} [netmask=string]
add attr {attr} {value} [attr=string] [value=string]
add bootaction [action=string] [args=string] [kernel=string] [ramdisk=string]
add distribution {distribution}
add firewall {category=index} [action=string] [chain=string] [network=string] [output-
    network=string] [protocol=string] [rulename=string] [rulesrc=string] [service=
    string]
add host {host} [cpus=int] [membership=string] [os=string] [rack=int] [rank=int]
add host alias {host} {name} [name=string]
add host attr {host} {attr} {value} [attr=string] [value=string]
add host bonded {host} [channel=string] [interfaces=string] [ip=string] [name=string] [
    network=string]
add host interface {host} {iface} [iface=string] [ip=string] [mac=string] [module=
    string] [name=string] [subnet=string] [vlan=string]
add host key {host} [key=string]
add host route {host} {address} {gateway} [netmask=string]
add host sec_attr {host} [attr=string] [crypted=boolean] [enc=string] [value=string]
add network {name} {subnet} {netmask} [dnszone=string] [mtu=string] [netmask=string] [
    servedns=boolean] [subnet=string]
add os attr {os} {attr} {value} [attr=string] [value=string]
add os route {os} {address} {gateway} [netmask=string]
add roll [roll]... [clean=bool]
add route {address} {gateway} [netmask=string]
add sec_attr {attr} [crypted=boolean] [enc=string] [value=string]
add var {service} {component} {value} [appliance=string] [component=string] [service=
    string] [value=string]
config host interface {host} [flag=string] [iface=string] [mac=string] [module=string]
create condor password [add=bool] [keyfile=string]
create distro [arch=string] [dist=string] [md5=bool] [rolls=string] [root=string] [
    version=string]
create keys [key=string] [passphrase=boolean]
create mirror {path} [arch=string] [rollname=string] [version=string]
create new roll {version} {name} {color} [color=string] [name=string] [version=string]
create package {directory} [prefix=string] [release=string] [version=string]
create roll {roll}...
disable roll {roll}... [arch=string] [version=string]
dump
dump appliance [appliance]...
dump appliance attr [appliance]
dump appliance route
dump attr
dump firewall
```

```
dump host [host]...
dump host attr [host]
dump host boot
dump host interface [host]...
dump host key
dump host roll
dump host route
dump network [network]...
dump os attr
dump os route
dump route
enable roll {roll}... [arch=string] [version=string]
help {command}
iterate host [host]... [command] [command=string]
list appliance [appliance]...
list appliance attr [appliance]
list appliance route
list appliance xml [appliance]...
list attr
list bootaction
list distribution [distribution]...
list firewall [category=index] [maxwidth=integer]
list help [subdir=string]
list host [host]...
list host alias [host]...
list host appliance [host]...
list host attr [host]
list host boot [host]...
list host firewall [host]... [maxwidth=integer]
list host graph [host]... [arch=string] [basedir=string]
list host installfile [section=string]
list host interface [host]...
list host key [host]...
list host macs {host} [key=string] [status=bool]
list host membership [host]...
list host partition [host]...
list host profile [host]...
list host roll [host]
list host route [host]
list host sec_attr {host}
list host xml [host]...
list license
list membership [membership]...
list network [network]...
list node xml [attrs=string] [basedir=string] [eval=bool] [gen=string] [missing-check=
    bool] [roll=string]
list os attr [os]
list os route
list roll [roll]...
list roll command [roll]...
list route
list sec_attr
list var [appliance=string] [component=string] [service=string]
open host console {host} [key=string] [vncflags=string]
remove appliance {name}
remove appliance attr {appliance} {attr} [attr=string]
remove appliance route {appliance} {address} [address=string]
remove attr {attr} [attr=string]
remove bootaction [action=string]
remove distribution {distribution}
remove firewall [category=index] {rulename}
remove host {host}...
remove host alias {host} {name} [name=string]
remove host attr {host} {attr} [attr=string]
remove host boot {host}...
remove host bootflags {host}...
remove host interface {host} {iface} [iface=string]
remove host key {host} [id=string]
remove host partition {host}... [partition=string]
```

```
remove host roll {host} {name} {version} {arch} [arch=string] [name=string] [os=string]
    [version=string]
remove host route {host} {address} [address=string]
remove host sec_attr {host} {attr}
remove network {network}...
remove os {os}
remove os attr {os} {attr} [attr=string]
remove os route {os} [address=string] [address=string]
remove roll {roll}... [arch=string] [version=string]
remove route {address} [address=string]
remove sec_attr {attr}
remove var {service} {component} [appliance=string] [component=string] [service=string]
report bug
report dbhost
report distro
report host
report host attr [host] [attr=string] [pydict=bool]
report host bootflags [host]...
report host condor config {host} [ConfigFile=string] [UIDdomain=string] [type=string]
report host condor interface {host} {subnet}
report host config411
report host dhcpd {host}
report host firewall {host}
report host ganglia gmond
report host interface {host} [iface=string]
report host network {host}
report host roll [host]
report host route {host}
report host sge config {host}
report knownhosts
report named
report post [arch=string] [attrs=string] [os=string]
report resolv
report resolv private
report script [arch=string] [attrs=string] [os=string]
report sge machines
report shosts
report tentakel
report version [major=boolean]
report zones
run host [host]... {command} [collate=string] [command=string] [delay=string] [managed=
    boolean] [num-threads=string] [stats=string] [timeout=string] [x11=boolean]
run host sec_attr {file}
run roll [roll]...
save host partitions
set appliance attr {appliance} {attr} {value} [attr=string] [value=string]
set attr {attr} {value} [attr=string] [value=string]
set host attr {host} {attr} {value} [attr=string] [value=string]
set host boot {host}... [action=string]
set host bootflags {host}... [flags=string]
set host comment {host}... {comment} [comment=string]
set host cpus {host}... {cpus} [cpus=string]
set host installaction {host}... {action} [action=string]
set host interface channel {host}... {iface} {channel} [channel=string] [iface=string]
set host interface iface {host}... {mac} {iface} [iface=string] [mac=string]
set host interface ip {host} {iface} {ip} [iface=string] [ip=string]
set host interface mac {host} {iface} {mac} [iface=string] [mac=string]
set host interface module {host}... {iface} {module} [iface=string] [module=string]
set host interface name {host} {iface} {name} [iface=string] [name=string]
set host interface options {host}... {iface} [iface=string] [options=string]
set host interface subnet {host}... {iface} {subnet} [iface=string] [subnet=string]
set host interface vlan {host}... {iface} {vlan} [iface=string] [vlan=string]
set host membership {host}... {membership} [membership=string]
set host name {host} {name} [name=string]
set host power {host}... [action=string] [key=string]
set host rack {host}... {rack} [rack=string]
set host rank {host}... {rank} [rank=string]
set host roll {host} {name} {version} {arch} [arch=string] [name=string] [os=string] [
    version=string]
```

```
set host runaction {host}... {action} [action=string]
set host sec_attr {host} [attr=string] [crypted=boolean] [enc=string] [value=string]
set network mtu {network}... {mtu} [mtu=string]
set network netmask {network}... {netmask} [netmask=string]
set network servedns {network} {servedns} [servedns=bool]
set network subnet {network}... {subnet} [subnet=string]
set network zone {network} {zone} [zone=string]
set os attr {os} {attr} {value} [attr=string] [value=string]
set sec_attr {attr} [crypted=boolean] [enc=string] [value=string]
set var {service} {component} {value} [appliance=string] [component=string] [service=
    string] [value=string]
swap host interface {host} [ifaces=string] [sync-config=boolean]
sync condor
sync config
sync dns
sync host condor [syncpassword=bool]
sync host firewall
sync host network
sync host sec_attr {host}
sync host sharedkey
sync users
update
```

8.2 Ejemplo mail monitorizacion

```
Message 17:
From root@cluster.torrex.org  Mon Aug  1 17:43:10 2016
Return-Path: <root@cluster.torrex.org>
X-Original-To: root
Delivered-To: root@cluster.torrex.org
To: root@cluster.torrex.org
From: logwatch@cluster.torrex.org
Subject: Logwatch for cluster.torrex.org (Linux)
Content-Type: text/plain; charset="iso-8859-1"
Date: Mon,  1 Aug 2016 17:43:06 +0200 (CEST)
Status: RO


##################### Logwatch 7.3.6 (05/19/07) ####################
Processing Initiated: Mon Aug  1 17:43:06 2016
Date Range Processed: yesterday
( 2016-Jul-31 )
Period is day.
Detail Level of Output: 0
Type of Output: unformatted
Logfiles for Host: cluster.torrex.org
################################################################################

-------------------- dhcpd Begin --------------------

DHCP Server Listening On:
LPF/eth0/00:14:85:1e:35:d7/10.1.0.0/16: 12 Time(s)

Unknown Entries:
Not searching LDAP since ldap-server, ldap-port and ldap-base-dn were not sp
ecified in the config file: 12 Time(s)


-------------------- dhcpd End --------------------


-------------------- httpd Begin --------------------


A total of 2 sites probed the server
```

```
10.1.255.251
10.1.255.253

A total of 1 possible successful probes were detected (the following URLs
contain strings that match one or more of a listing of strings that
indicate a possible exploit):

/411.d/etc.passwd HTTP Response 200

------------------------ httpd End ----------------------------


------------------------ Init Begin ---------------------------


**Unmatched Entries**
Disconnected from system bus
Disconnected from system bus
Disconnected from system bus
Disconnected from system bus

------------------------ Init End -----------------------------


------------------------ Kernel Begin -------------------------


WARNING:  Kernel Errors Present
usb 2-2: can't set config #1, error -71 ...:  1 Time(s)
usb 2-2: device descriptor read/64, error -71 ...:  1 Time(s)

------------------------ Kernel End ---------------------------


------------------------ Named Begin --------------------------


Received control channel commands
reload: 11 Time(s)
stop: 1 Time(s)

**Unmatched Entries**
------------------------------------------------------------: 2 Time(s)
BIND 9 is maintained by Internet Systems Consortium,: 1 Time(s)
Inc. (ISC), a non-profit 501(c)(3) public-benefit : 1 Time(s)
Warning: 'empty-zones-enable/disable-empty-zone' not set: disabling RFC 1918
empty zones: 12 Time(s)
available at https://www.isc.org/support: 1 Time(s)
corporation.  Support and training for BIND 9 are : 1 Time(s)
error (network unreachable) resolving 'compute-0-2.local.torrex.org/A/IN': 2
001:500:48::1#53: 1 Time(s)
error (unexpected RCODE REFUSED) resolving 'comnpute-0-4.torrex.org/A/IN': 1
85.23.17.147#53: 1 Time(s)
error (unexpected RCODE REFUSED) resolving 'comnpute-0-4.torrex.org/A/IN': 6
2.75.203.246#53: 1 Time(s)
error (unexpected RCODE REFUSED) resolving 'comnpute-0-4.torrex.org/AAAA/IN'
: 185.23.17.147#53: 1 Time(s)
error (unexpected RCODE REFUSED) resolving 'comnpute-0-4.torrex.org/AAAA/IN'
: 62.75.203.246#53: 1 Time(s)
error (unexpected RCODE REFUSED) resolving 'compute-0-0.torrex.org/A/IN': 18
5.23.17.147#53: 28 Time(s)
error (unexpected RCODE REFUSED) resolving 'compute-0-0.torrex.org/A/IN': 62
.75.203.246#53: 28 Time(s)
error (unexpected RCODE REFUSED) resolving 'compute-0-2.local.torrex.org/A/I
N': 185.23.17.147#53: 28 Time(s)
error (unexpected RCODE REFUSED) resolving 'compute-0-2.local.torrex.org/A/I
N': 62.75.203.246#53: 28 Time(s)
error (unexpected RCODE REFUSED) resolving 'compute-0-2.torrex.org/A/IN': 18
5.23.17.147#53: 3 Time(s)
```

```
error (unexpected RCODE REFUSED) resolving 'compute-0-2.torrex.org/A/IN': 62
.75.203.246#53: 3 Time(s)
error (unexpected RCODE REFUSED) resolving 'compute-0-2.torrex.org/AAAA/IN':
185.23.17.147#53: 1 Time(s)
error (unexpected RCODE REFUSED) resolving 'compute-0-2.torrex.org/AAAA/IN':
62.75.203.246#53: 1 Time(s)
error (unexpected RCODE REFUSED) resolving 'compute-0-4.torrex.org/A/IN': 18
5.23.17.147#53: 2 Time(s)
error (unexpected RCODE REFUSED) resolving 'compute-0-4.torrex.org/A/IN': 62
.75.203.246#53: 2 Time(s)
error (unexpected RCODE REFUSED) resolving 'ganglia.torrex.org/A/IN': 185.23
.17.147#53: 2 Time(s)
error (unexpected RCODE REFUSED) resolving 'ganglia.torrex.org/A/IN': 62.75.
203.246#53: 2 Time(s)
error (unexpected RCODE REFUSED) resolving 'ganglia.torrex.org/AAAA/IN': 185
.23.17.147#53: 2 Time(s)
error (unexpected RCODE REFUSED) resolving 'ganglia.torrex.org/AAAA/IN': 62.
75.203.246#53: 2 Time(s)
generating session key for dynamic DNS: 1 Time(s)
set up managed keys zone for view _default, file 'dynamic/managed-keys.bind'
: 1 Time(s)
sizing zone task pool based on 9 zones: 12 Time(s)
zone local/IN: local/MX 'mail.local' has no address records (A or AAAA): 12
Time(s)

---------- Named End ----------


---------- pam_unix Begin ----------

gdm-password:
Unknown Entries:
session opened for user zxcv90 by (uid=0): 1 Time(s)

su:
Sessions Opened:
root -> root: 4 Time(s)

sudo:
Authentication Failures:
zxcv90(0) -> zxcv90: 1 Time(s)
Unknown Entries:
auth could not identify password for [zxcv90]: 1 Time(s)
conversation failed: 1 Time(s)


---------- pam_unix End ----------


---------- Postfix Begin ----------

1    *Warning: Database file needs update

58.563K  Bytes accepted                                59,969
8.999K   Bytes delivered                               9,215
=======  ============================================================

7    Accepted                                  100.00%
-------  ------------------------------------------------------------
7    Total                                     100.00%
=======  ============================================================

2    Connections made
2    Disconnections
7    Removed from queue
4    Delivered
2    Sent via SMTP
1    Bounce (local)
1    DSNs undeliverable
```

```
4    Postfix start
4    Postfix stop



**Unmatched Entries**
1    Jul 31 17:12:38 cluster postfix/postfix-script[2073]: warning: /usr/
lib/sendmail and /usr/sbin/sendmail differ
1    Jul 31 17:26:28 compute-0-1 postfix/postfix-script[1438]: warning: /
usr/lib/sendmail and /usr/sbin/sendmail differ
1    Jul 31 17:13:13 compute-0-3 postfix/postfix-script[1487]: warning: /
usr/lib/sendmail and /usr/sbin/sendmail differ
1    Jul 31 18:36:51 compute-0-3 postfix/postfix-script[1479]: warning: R
eplace one by a symbolic link to the other
1    Jul 31 17:26:29 compute-0-1 postfix/postfix-script[1439]: warning: R
eplace one by a symbolic link to the other
1    Jul 31 17:12:38 cluster postfix/postfix-script[2074]: warning: Repla
ce one by a symbolic link to the other
1    Jul 31 17:13:13 compute-0-3 postfix/postfix-script[1488]: warning: R
eplace one by a symbolic link to the other
1    Jul 31 18:36:51 compute-0-3 postfix/postfix-script[1478]: warning: /
usr/lib/sendmail and /usr/sbin/sendmail differ

---------- Postfix End ----------


---------- Connections (secure-log) Begin ----------


**Unmatched Entries**
polkitd(authority=local): Registered Authentication Agent for session /org/f
reedesktop/ConsoleKit/Session1 (system bus name :1.19 [/usr/libexec/polkit-gnome
-authentication-agent-1], object path /org/gnome/PolicyKit1/AuthenticationAgent,
locale en_US.iso885915): 1 Time(s)
polkitd(authority=local): Registered Authentication Agent for session /org/f
reedesktop/ConsoleKit/Session2 (system bus name :1.29 [/usr/libexec/polkit-gnome
-authentication-agent-1], object path /org/gnome/PolicyKit1/AuthenticationAgent,
locale en_US.iso885915): 1 Time(s)
polkitd(authority=local): Unregistered Authentication Agent for session /org
/freedesktop/ConsoleKit/Session1 (system bus name :1.19, object path /org/gnome/
PolicyKit1/AuthenticationAgent, locale en_US.iso885915) (disconnected from bus):
1 Time(s)
polkitd(authority=local): Unregistered Authentication Agent for session /org
/freedesktop/ConsoleKit/Session2 (system bus name :1.29, object path /org/gnome/
PolicyKit1/AuthenticationAgent, locale en_US.iso885915) (disconnected from bus):
1 Time(s)

---------- Connections (secure-log) End ----------


---------- SSHD Begin ----------


SSHD Killed: 4 Time(s)

SSHD Started: 5 Time(s)

Users logging in through sshd:
root:
10.1.1.1 (cluster.local): 14 times
zxcv90:
10.1.1.1 (cluster.local): 8 times


Received disconnect:
11: disconnected by user : 22 Time(s)

---------- SSHD End ----------
```

```
------------------------ Sudo (secure-log) Begin ------------------------

========================================================================

zxcv90 => root
-------------
/opt/rocks/bin/rocks - 2 Times.


**Unmatched Entries**
pam_unix(sudo:auth): conversation failed: 1 Time(s)
pam_unix(sudo:auth): auth could not identify password for [zxcv90]: 1 Time(s
)
------------------------ Sudo (secure-log) End --------------------------


------------------------ XNTPD Begin ------------------------


XNTPD Killed: 4 Time(s)

XNTPD Started: 4 Time(s)

Time Reset 4 times (total: -4.979876 s  average: -1.244969 s)

Total synchronizations 9 (hosts: 3)

**Unmatched Entries**
Listening on routing socket on fd #24 for interface updates: 1 time(s)
Listening on routing socket on fd #19 for interface updates: 3 time(s)

------------------------ XNTPD End --------------------------


------------------------ Disk Space Begin ------------------------

Filesystem            Size  Used Avail Use% Mounted on
/dev/sda1             146G  8.0G  131G   6% /
/dev/sdb1             7.3G  1.4G  5.9G  19% /media/KERNEL DISK


------------------------ Disk Space End --------------------------


########################## Logwatch End ##############################
```

8.3 Fichero make.def para configuracion de suite NAS Parallel Benchmarks

```
#---------------------------------------------------------------------------
#
#                    SITE- AND/OR PLATFORM-SPECIFIC DEFINITIONS.
#
#---------------------------------------------------------------------------

#---------------------------------------------------------------------------
# Items in this file will need to be changed for each platform.
# (Note these definitions are inconsistent with NPB2.1.)
#---------------------------------------------------------------------------

#---------------------------------------------------------------------------
# Parallel Fortran:
```

```
#
# For CG, EP, FT, MG, LU, SP and BT, which are in Fortran, the following must
# be defined:
#
# MPIF77      - Fortran compiler
# FFLAGS      - Fortran compilation arguments
# FMPI_INC    - any -I arguments required for compiling MPI/Fortran
# FLINK       - Fortran linker
# FLINKFLAGS - Fortran linker arguments
# FMPI_LIB    - any -L and -l arguments required for linking MPI/Fortran
#
# compilations are done with $(MPIF77) $(FMPI_INC) $(FFLAGS) or
#                            $(MPIF77) $(FFLAGS)
# linking is done with       $(FLINK) $(FMPI_LIB) $(FLINKFLAGS)
#---------------------------------------------------------------------

#---------------------------------------------------------------------
# This is the fortran compiler used for MPI programs
#---------------------------------------------------------------------
MPIF77 = mpif77
# This links MPI fortran programs; usually the same as ${MPIF77}
FLINK = $(MPIF77)

#---------------------------------------------------------------------
# These macros are passed to the linker to help link with MPI correctly
#---------------------------------------------------------------------
FMPI_LIB  =

#---------------------------------------------------------------------
# These macros are passed to the compiler to help find 'mpif.h'
#---------------------------------------------------------------------
FMPI_INC =

#---------------------------------------------------------------------
# Global *compile time* flags for Fortran programs
#---------------------------------------------------------------------
FFLAGS  =
# FFLAGS =

#---------------------------------------------------------------------
# Global *link time* flags. Flags for increasing maximum executable
# size usually go here.
#---------------------------------------------------------------------
FLINKFLAGS =


#---------------------------------------------------------------------
# Parallel C:
#
# For IS, which is in C, the following must be defined:
#
# MPICC       - C compiler
# CFLAGS      - C compilation arguments
# CMPI_INC    - any -I arguments required for compiling MPI/C
# CLINK       - C linker
# CLINKFLAGS - C linker flags
# CMPI_LIB    - any -L and -l arguments required for linking MPI/C
#
# compilations are done with $(MPICC) $(CMPI_INC) $(CFLAGS) or
#                            $(MPICC) $(CFLAGS)
# linking is done with       $(CLINK) $(CMPI_LIB) $(CLINKFLAGS)
#---------------------------------------------------------------------

#---------------------------------------------------------------------
# This is the C compiler used for MPI programs
#---------------------------------------------------------------------
MPICC = mpicc
# This links MPI C programs; usually the same as ${MPICC}
CLINK = $(MPICC)
```

```
#---------------------------------------------------------------------------
# These macros are passed to the linker to help link with MPI correctly
#---------------------------------------------------------------------------
CMPI_LIB  =

#---------------------------------------------------------------------------
# These macros are passed to the compiler to help find 'mpi.h'
#---------------------------------------------------------------------------
CMPI_INC =

#---------------------------------------------------------------------------
# Global *compile time* flags for C programs
#---------------------------------------------------------------------------
CFLAGS  = -march=pentium4 -O2
# CFLAGS =

#---------------------------------------------------------------------------
# Global *link time* flags. Flags for increasing maximum executable
# size usually go here.
#---------------------------------------------------------------------------
CLINKFLAGS =


#---------------------------------------------------------------------------
# MPI dummy library:
#
# Uncomment if you want to use the MPI dummy library supplied by NAS instead
# of the true message-passing library. The include file redefines several of
# the above macros. It also invokes make in subdirectory MPI_dummy. Make
# sure that no spaces or tabs precede include.
#---------------------------------------------------------------------------
# include ../config/make.dummy


#---------------------------------------------------------------------------
# Utilities C:
#
# This is the C compiler used to compile C utilities.  Flags required by
# this compiler go here also; typically there are few flags required; hence
# there are no separate macros provided for such flags.
#---------------------------------------------------------------------------
CC  = cc -g


#---------------------------------------------------------------------------
# Destination of executables, relative to subdirs of the main directory. .
#---------------------------------------------------------------------------
BINDIR  = ../bin


#---------------------------------------------------------------------------
# Some machines (e.g. Crays) have 128-bit DOUBLE PRECISION numbers, which
# is twice the precision required for the NPB suite. A compiler flag
# (e.g. -dp) can usually be used to change DOUBLE PRECISION variables to
# 64 bits, but the MPI library may continue to send 128 bits. Short of
# recompiling MPI, the solution is to use MPI_REAL to send these 64-bit
# numbers, and MPI_COMPLEX to send their complex counterparts. Uncomment
# the following line to enable this substitution.
#
# NOTE:  IF THE I/O BENCHMARK IS BEING BUILT, WE USE CONVERTFLAG TO
#        SPECIFIY THE FORTRAN RECORD LENGTH UNIT. IT IS A SYSTEM-SPECIFIC
#        VALUE (USUALLY 1 OR 4). UNCOMMENT THE SECOND LINE AND SUBSTITUTE
#        THE CORRECT VALUE FOR "length".
#        IF BOTH 128-BIT DOUBLE PRECISION NUMBERS AND I/O ARE TO BE ENABLED,
#        UNCOMMENT THE THIRD LINE AND SUBSTITUTE THE CORRECT VALUE FOR
#        "length"
#---------------------------------------------------------------------------
# CONVERTFLAG = -DCONVERTDOUBLE
```

```
CONVERTFLAG = -DFORTRAN_REC_SIZE=1
# CONVERTFLAG = -DCONVERTDOUBLE -DFORTRAN_REC_SIZE=length


#-----------------------------------------------------------------------
# The variable RAND controls which random number generator
# is used. It is described in detail in Doc/README.install.
# Use "randi8" unless there is a reason to use another one.
# Other allowed values are "randi8_safe", "randdp" and "randdpvec"
#-----------------------------------------------------------------------
RAND   = randi8
# The following is highly reliable but may be slow:
# RAND   = randdp
```

8.4 Contenido de un fichero de trazas generado con Extrae: libmpi_f77.so.1.0.3.prv

```
#Paraver (16/11/2016 at 18:59):
284274975514_ns:4(1,1,1,1):1:4(1:1,1:2,1:3,1:4),5
c:1:1:4:1:2:3:4
c:1:2:1:1
c:1:3:1:2
c:1:4:1:3
c:1:5:1:4
1:1:1:1:1:0:2491838:2
1:2:1:2:1:0:3792814:2
2:3:1:3:1:0:40000001:1:40000050:245821306675:40000133:0
1:3:1:3:1:0:44818526:1
2:3:1:3:1:0:40000018:1:41999999:1:42000050:0:42000059:0:42000055:0:42000046:0
1:4:1:4:1:0:457570:2
2:4:1:4:1:457570:40000001:1:40000050:239417873858:40000133:0
1:4:1:4:1:457570:45069574:1
2:4:1:4:1:457570:40000018:1:41999999:1:42000050:0:42000059:0:42000055:0:42000046:0
2:1:1:1:1:2491838:40000001:1:40000050:613028201550:40000133:0
1:1:1:1:1:2491838:45836387:1
2:1:1:1:1:2491838:40000018:1:41999999:1:42000050:0:42000059:0:42000055:0:42000046:0
2:2:1:2:1:3792814:40000001:1:40000050:444460577196:40000133:0
1:2:1:2:1:3792814:44679377:1
2:2:1:2:1:3792814:40000018:1:41999999:1:42000050:0:42000059:0:42000055:0:42000046:0
1:2:1:2:1:44679377:46263958:15
2:2:1:2:1:44679377:50000003:31:40000036:1851:40000037:0:40000038:1:42000050:4627507:
42000059:20941740:42000055:1086465:42000046:4294967296:45000000:0:45000001:0:45000006:0:

45000007:0:45000014:0:45000015:0
1:3:1:3:1:44818526:46263958:15
2:3:1:3:1:44818526:50000003:31:40000036:1920:40000037:0:40000038:1:42000050:17183414:
42000059:49774873:42000055:4024560:42000046:4294967296:45000000:0:45000001:0:45000006:0:

45000007:0:45000014:0:45000015:0
1:4:1:4:1:45069574:46263958:15
2:4:1:4:1:45069574:50000003:31:40000036:1827:40000037:0:40000038:1:42000050:6053183:
42000059:24098742:42000055:1432021:42000046:4294967296:45000000:0:45000001:0:45000006:0:

45000007:0:45000014:0:45000015:0
1:1:1:1:1:45836387:46263958:15
2:1:1:1:1:45836387:50000003:31:40000036:1860:40000037:0:40000038:1:42000050:2514837:
42000059:13214252:42000055:571775:42000046:4294967296:45000000:0:45000001:0:45000006:0:
45000007:0:45000014:0:45000015:0
1:1:1:1:1:46263958:46519435:1
2:1:1:1:1:46263958:50000003:0:42000050:6317:42000059:67314:42000055:1483:42000046:
52342400285828520
1:2:1:2:1:46263958:46473986:1
2:2:1:2:1:46263958:50000003:0:42000050:6326:42000059:63582:42000055:1482:42000046:
46343464844357256
1:3:1:3:1:46263958:46390389:1
```

```
2:3:1:3:1:46263958:50000003:0:42000050:6219:42000059:54797:42000055:1453:42000046:
48841555262167800
1:4:1:4:1:46263958:46405432:1
2:4:1:4:1:46263958:50000003:0:42000050:6356:42000059:58553:42000055:1492:42000046:
39658434149473976
1:3:1:3:1:46390389:59440127:15
2:3:1:3:1:46390389:50000003:19:42000050:30677:42000059:177649:42000055:6305:70000001:1:
80000001:1:70000002:1:80000002:1:70000003:2:80000003:2
1:4:1:4:1:46405432:73288359:15
2:4:1:4:1:46405432:50000003:19:42000050:30921:42000059:212913:42000055:6374:70000001:1:
80000001:1:70000002:1:80000002:1:70000003:2:80000003:2
1:2:1:2:1:46473986:56898416:15
2:2:1:2:1:46473986:50000003:19:42000050:30772:42000059:253330:42000055:6336:70000001:1:
80000001:1:70000002:1:80000002:1:70000003:2:80000003:2
1:1:1:1:1:46519435:62700208:15
2:1:1:1:1:46519435:50000003:19:42000050:33139:42000059:283854:42000055:7075:70000001:1:
80000001:1:70000002:1:80000002:1:70000003:2:80000003:2
1:2:1:2:1:56898416:56946948:1
2:2:1:2:1:56898416:50000003:0:42000050:162746:42000059:1140124:42000055:35311
1:2:1:2:1:56946948:57029451:15
```

8.5 Salida Kernel MG - Class C - Ejecucion en 4 nodos

```
 NAS Parallel Benchmarks 3.3 -- MG Benchmark

 No input file. Using compiled defaults
 Size:  512x 512x 512  (class C)
 Iterations:   20
 Number of processes:     4

 Initialization time:          24.513 seconds

 iter    1
 iter    5
 iter   10
 iter   15
 iter   20

 Benchmark completed
 VERIFICATION SUCCESSFUL
 L2 Norm is  0.5706732285736E-06
 Error is    0.6577169840085E-12


 MG Benchmark Completed.
 Class           =                        C
 Size            =            512x 512x 512
 Iterations      =                       20
 Time in seconds =                   239.84
 Total processes =                        4
 Compiled procs  =                        4
 Mop/s total     =                   649.16
 Mop/s/process   =                   162.29
 Operation type  =          floating point
 Verification    =              SUCCESSFUL
 Version         =                    3.3.1
 Compile date    =              29 Oct 2016

 Compile options:
 MPIF77      = mpif77
 FLINK       = $(MPIF77)
 FMPI_LIB    = (none)
 FMPI_INC    = (none)
 FFLAGS      = (none)
 FLINKFLAGS  = (none)
 RAND        = randi8
```

```
 Please send feedbacks and/or the results of this run to:
NPB Development Team
Internet: npb@nas.nasa.gov
```

8.6 Salida Kernel IS - Class C - Ejecucion en 4 nodos

```
NAS Parallel Benchmarks 3.3 -- IS Benchmark

Size:  134217728  (class C)
Iterations:   10
Number of processes:     4

iteration
1
2
3
4
5
6
7
8
9
10


IS Benchmark Completed
Class             =                        C
Size              =                134217728
Iterations   =                       10
Time in seconds =                    143.73
Total processes =                         4
Compiled procs  =                         4
Mop/s total     =                      9.34
Mop/s/process   =                      2.33
Operation type  =              keys ranked
Verification    =               SUCCESSFUL
Version         =                    3.3.1
Compile date    =              19 Oct 2016

Compile options:
MPICC        = mpicc
CLINK        = $(MPICC)
CMPI_LIB     = (none)
CMPI_INC     = (none)
CFLAGS   = (none)
CLINKFLAGS   = (none)


Please send feedbacks and/or the results of this run to:

NPB Development Team
npb@nas.nasa.gov
```

8.7 Salida Kernel CG - Class C - Ejecucion en 4 nodos

```
NAS Parallel Benchmarks 3.3 -- CG Benchmark

Size:       75000
Iterations:    75
Number of active processes:     4
Number of nonzeroes per row:       13
Eigenvalue shift: .600E+02

iteration           ||r||                 zeta
1 0.22436181736246E-12    59.9994751578754
2 0.87419521699280E-15    21.7627846142534
3 0.91026104494405E-15    22.2876617043224
4 0.93487388008893E-15    22.5230738188351
5 0.94556567772461E-15    22.6275390653893
6 0.94513061899606E-15    22.6740259189539
7 0.95496057751994E-15    22.6949056826254
8 0.95208777792337E-15    22.7044023166870
9 0.95045949920555E-15    22.7087834345616
10  0.95771303118780E-15    22.7108351397173
11  0.95265861967217E-15    22.7118107121337
12  0.95510027111077E-15    22.7122816240974
13  0.95704610017487E-15    22.7125122663251
14  0.95184692953200E-15    22.7126268007600
15  0.95440392536245E-15    22.7126844161815
16  0.95426689529620E-15    22.7127137461757
17  0.94760787195652E-15    22.7127288401998
18  0.95251112100049E-15    22.7127366848299
19  0.95243934803094E-15    22.7127407981220
20  0.94851258088846E-15    22.7127429721363
21  0.95209211307528E-15    22.7127441294025
22  0.94937329942693E-15    22.7127447493899
23  0.94832692134681E-15    22.7127450834529
24  0.95329012913226E-15    22.7127452643880
25  0.95057279938431E-15    22.7127453628459
26  0.94892755432761E-15    22.7127454166512
27  0.94434492079657E-15    22.7127454461693
28  0.94976883930005E-15    22.7127454624206
29  0.95706261734286E-15    22.7127454713970
30  0.94329872831642E-15    22.7127454763706
31  0.94667019756029E-15    22.7127454791340
32  0.94829493007511E-15    22.7127454806733
33  0.95297844810828E-15    22.7127454815324
34  0.94608002525005E-15    22.7127454820135
35  0.95390526241480E-15    22.7127454822838
36  0.94561018393403E-15    22.7127454824349
37  0.94406423934563E-15    22.7127454825207
38  0.94471352744700E-15    22.7127454825686
39  0.94658752058174E-15    22.7127454825961
40  0.95125237653647E-15    22.7127454826112
41  0.94977897687140E-15    22.7127454826204
42  0.94469546451062E-15    22.7127454826251
43  0.94248688231054E-15    22.7127454826280
44  0.94822552769104E-15    22.7127454826296
45  0.94269210624200E-15    22.7127454826306
46  0.94087794508176E-15    22.7127454826307
47  0.94120529555199E-15    22.7127454826314
48  0.94762699191791E-15    22.7127454826317
49  0.94297387043967E-15    22.7127454826316
50  0.94392468865383E-15    22.7127454826318
51  0.93365870411207E-15    22.7127454826316
52  0.94152545714380E-15    22.7127454826318
53  0.94252150194625E-15    22.7127454826321
54  0.93862339410146E-15    22.7127454826315
55  0.93627085310765E-15    22.7127454826320
56  0.93938043822467E-15    22.7127454826319
57  0.93721303193366E-15    22.7127454826315
58  0.93908348644166E-15    22.7127454826317
```

```
59   0.93691506150337E-15     22.7127454826315
60   0.93933037856209E-15     22.7127454826318
61   0.93665925026070E-15     22.7127454826316
62   0.93864368093939E-15     22.7127454826318
63   0.93644008482771E-15     22.7127454826314
64   0.93705413624994E-15     22.7127454826317
65   0.93923709171489E-15     22.7127454826315
66   0.93067049174430E-15     22.7127454826315
67   0.93733784173068E-15     22.7127454826317
68   0.92998500616442E-15     22.7127454826318
69   0.93181291636231E-15     22.7127454826318
70   0.94416749763259E-15     22.7127454826317
71   0.93942106870259E-15     22.7127454826315
72   0.93526157970142E-15     22.7127454826314
73   0.92948672086701E-15     22.7127454826318
74   0.93597252739281E-15     22.7127454826317
75   0.93627128900379E-15     22.7127454826317
Benchmark completed
VERIFICATION SUCCESSFUL
Zeta is      0.2271274548263E+02
Error is     0.3222239329862E-13


CG Benchmark Completed.
Class           =                        B
Size            =                    75000
Iterations      =                       75
Time in seconds =                   261.08
Total processes =                        4
Compiled procs  =                        4
Mop/s total     =                   209.55
Mop/s/process   =                    52.39
Operation type  =          floating point
Verification    =              SUCCESSFUL
Version         =                    3.3.1
Compile date    =              29 Oct 2016

Compile options:
MPIF77      = mpif77
FLINK       = $(MPIF77)
FMPI_LIB    = (none)
FMPI_INC    = (none)
FFLAGS      = (none)
FLINKFLAGS  = (none)
RAND        = randi8


Please send feedbacks and/or the results of this run to:

NPB Development Team
Internet: npb@nas.nasa.gov
```

8.8 Salida Kernel FT - Class C - Ejecucion en 4 nodos

```
NAS Parallel Benchmarks 3.3 -- FT Benchmark

No input file inputft.data. Using compiled defaults
Size                :  512x 256x 256
Iterations          :              20
Number of processes :               4
Processor array     :          1x   4
Layout type         :              1D
Initialization time =   20.561954021453857
T =     1     Checksum =     5.177643571579D+02     5.077803458597D+02
T =     2     Checksum =     5.154521291263D+02     5.088249431599D+02
T =     3     Checksum =     5.146409228650D+02     5.096208912659D+02
```

```
T =    4     Checksum =    5.142378756213D+02    5.101023387619D+02
T =    5     Checksum =    5.139626667737D+02    5.103976610618D+02
T =    6     Checksum =    5.137423460082D+02    5.105948019802D+02
T =    7     Checksum =    5.135547056878D+02    5.107404165783D+02
T =    8     Checksum =    5.133910925467D+02    5.108576573661D+02
T =    9     Checksum =    5.132470705390D+02    5.109577278523D+02
T =   10     Checksum =    5.131197729984D+02    5.110460304483D+02
T =   11     Checksum =    5.130070319283D+02    5.111252433800D+02
T =   12     Checksum =    5.129070537032D+02    5.111968077719D+02
T =   13     Checksum =    5.128182883503D+02    5.112616233064D+02
T =   14     Checksum =    5.127393733383D+02    5.113203605551D+02
T =   15     Checksum =    5.126691062021D+02    5.113735928093D+02
T =   16     Checksum =    5.126064276005D+02    5.114218460548D+02
T =   17     Checksum =    5.125504076570D+02    5.114656139760D+02
T =   18     Checksum =    5.125002331721D+02    5.115053595966D+02
T =   19     Checksum =    5.124551951846D+02    5.115415130407D+02
T =   20     Checksum =    5.124146770029D+02    5.115744692211D+02
Result verification successful
class = B


FT Benchmark Completed.
Class           =                        B
Size            =            512x 256x 256
Iterations      =                       20
Time in seconds =                   400.13
Total processes =                        4
Compiled procs  =                        4
Mop/s total     =                   230.06
Mop/s/process   =                    57.51
Operation type  =             floating point
Verification    =                 SUCCESSFUL
Version         =                    3.3.1
Compile date    =              29 Oct 2016
```

8.9 Salida Kernel EP - Class C - Ejecucion en 4 nodos

```
NAS Parallel Benchmarks 3.3 -- EP Benchmark

Number of random numbers generated:   8589934592
Number of active processes:                    4

EP Benchmark Results:

CPU Time =  282.7338
N = 2^   32
No. Gaussian Pairs =    3373275903.
Sums =     4.764367927996160D+04   -8.084072988039313D+04
Counts:
0     1572172634.
1     1501108549.
2      281805648.
3    17761221.
4         424017.
5           3821.
6             13.
7              0.
8              0.
9              0.


EP Benchmark Completed.
Class           =                        C
Size            =               8589934592
Iterations      =                        0
Time in seconds =                   282.73
```

```
Total processes =                         4
Compiled procs  =                         4
Mop/s total     =                     30.38
Mop/s/process   =                      7.60
Operation type  = Random numbers generated
Verification    =                SUCCESSFUL
Version         =                     3.3.1
Compile date    =               19 Oct 2016

Compile options:
MPIF77      = mpif77
FLINK       = $(MPIF77)
FMPI_LIB    = (none)
FMPI_INC    = (none)
FFLAGS      = (none)
FLINKFLAGS  = (none)
RAND        = randi8
```

8.10 Scripts de lanzamiento de trabajos

```
#!/bin/bash
#
#$ -cwd
#$ -j y
#$ -S /bin/bash
#
#MPI_DIR=/opt/mpich/gnu/
#HPL_DIR=/opt/hpl/mpich-hpl/

#Parametro 1: Numero de procesos
$1

#Parametro 2: Clase
$2

mpirun -np $1 -machinefile ../machines/mach$1 ../bin/CG/CLASS_$2/cg.$2.$1 >> CG/cg.$2.
    $1.out
```

```
#!/bin/bash
#
#$ -cwd
#$ -j y
#$ -S /bin/bash
#
#MPI_DIR=/opt/mpich/gnu/
#HPL_DIR=/opt/hpl/mpich-hpl/

#Parametro 1: Numero de procesos
$1

#Parametro 2: Clase
$2

mpirun -np $1 -machinefile ../machines/mach$1 ../bin/EP/CLASS_$2/ep.$2.$1 >> EP/ep.$2.
    $1.out
```

```
#!/bin/bash
#
#$ -cwd
#$ -j y
#$ -S /bin/bash
#
#MPI_DIR=/opt/mpich/gnu/
#HPL_DIR=/opt/hpl/mpich-hpl/
```

```
#Parametro 1: Numero de procesos
$1

#Parametro 2: Classe
$2

mpirun -np $1 -machinefile ../machines/mach$1 ../bin/FT/CLASS_$2/ft.$2.$1 >> FT/ft.$2.
    $1.out
```

8.11 Salida ejecucion PRIME MPI en 2 nodos

```
14 September 2016 05:12:01 PM

PRIME_MPI
C/MPI version

An MPI example program to count the number of primes.
The number of processes is 2

N            Pi       Time

1            0        0.001476
2            1        0.000110
4            2        0.000101
8            4        0.000193
16           6        0.000112
32           11       0.000105
64           18       0.000119
128          31       0.000128
256          54       0.000195
512          97       0.000442
1024         172      0.001279
2048         309      0.004253
4096         564      0.015311
8192         1028     0.055366
16384        1900     0.211747
32768        3512     0.781306
65536        6542     2.831970
131072       12251    10.558758
262144       23000    39.426327
524288       43390    149.005925
1048576      82025    561.893174

PRIME_MPI - Master process:
Normal end of execution.

14 September 2016 05:24:45 PM

real    12m45.327s
user    0m0.082s
sys     0m0.045s
```

Bibliografía

[1] Fernando Pardo Carpio (2002)
Arquitecturas Avanzadas,
Universitat Valencia

[2] Joseph D Sloan (November 2004)
High Performance Linux Clusters with OSCAR, Rocks, OpenMosix, and MPI - A Comprehensive Getting-Started Guide
O'Reilly Media

[3] Hesam El-Rewini, Mostafa Abd-El-Barr (2005)
Advanced Computer Architecture And Parallel Processing
Willey.

[4] `https://wikis.nyu.edu/display/NYUHPC/High+Performance+Computing+at+NYU`
High Performance Computing at NYU, (2016)
Created by Meredith Rendall, last modified by Shenglong Wang

[5] D.Bailey,E.Barszcz,J.Barton,D.Browning,R.Carter,L.Dagum,
R.Fatoohi,S.Fineber,P.Frederikson,T.Lasinski,R.Schreiber,
H,Simon,V.Venkatakrishnan,S.Weeratunga (March 1994)
The Nas Parallel Benchmarks
RNR Technical Report RNR-94-007

[6] Parkson Wong, Rob F. Van der Wijngaart (January 2003)
NAS Parallel Benchmarks I/O Version 2.4
NAS Technical Report NAS-03-002, NASA Advanced Supercomputing Division

[7] Huiyu Feng, Rob F. Van der Wijngaart, Rupak Biswas, Catherine Mavriplis, (July 2006)
Unstructured Adaptive (UA) NAS Parallel Benchmark, Version1.0
NAS Technical Report NAS-04-006, NASA Advanced Supercomputing Division

[8] Brian Wylie (October 2010)
NPB3.3-MPI/BT tutorial example application
Jülich Supercomputing Centre

[9] `https://www.nas.nasa.gov/publications/npb_problem_sizes.html`
Problem Sizes and Parameters in NAS Parallel Benchmarks, (2016)
NASA Advanced Supercomputing Division

[10] `https://gcc.gnu.org/onlinedocs/gcc/Optimize-Options.html`
Optimize Options - Using the GNU Compiler Collection (GCC), (2016)
Free Software Foundation

[11] `https://wiki.gentoo.org/wiki/GCC_optimization`
Gento Wiki, nightmorph, (2016)
GCC optimization

[12] `https://www.nas.nasa.gov/publications/npb.html`
NAS Parallel Benchmarks, (2016)
NASA Advanced Supercomputing Division

[13] *Extrae - User guide manual for version 3.2.1*
BSC - Tools manuals
Barcelona Supercomputing Center (2015)

[14] *Paraver (Reference) - Paraver Program Visualization and Analysis Tool -*
BSC - Tools manuals
Barcelona Supercomputing Center (2001)

[15] *Paraver (Tutorial)- Paraver Program Visualization and Analysis Tool -*
BSC - Tools manuals
Barcelona Supercomputing Center (2001)

[16] *Paraver (Tracefile Description)- Paraver Program Visualization and Analysis Tool -*
BSC - Tools manuals
Barcelona Supercomputing Center (2001)