Computación de alto rendimiento sobre un entorno de virtualización basado en contenedores

Daniel Domínguez San Segundo

Cátedra UAM-IBM
Escuela Politécnica Superior,
Universidad Autónoma de Madrid,
Calle Francisco Tomás y Valiente, 11, 28049 Madrid, España
daniel.dominguezs@estudiante.uam.es
http://www.catedrauamibm.com

Resumen: El rendimiento de un sistema es un aspecto crítico en la clusterización y especialmente en el ámbito de HPC (*High Performance Computing* – Computación de Alto Rendimiento). Por otra parte, la virtualización es una tecnología que cada día está ganando presencia en importantes escenarios tradicionalmente ligados a HPC. Entre estos escenarios se encuentran la alta disponibilidad de recursos proporcionados por servidores, o la oferta de infraestructuras, plataformas y software como servicios en la nube.

En este estudio se ha comprobado por medio del *benchmark* de referencia HPL (*High Performance Linpack*) las diferencias de rendimiento que hay al utilizar una misma infraestructura física para un clúster que se utilice de forma nativa, o como infraestructura física para conformar un clúster virtual equivalente en recursos totales.

Aunque la virtualización supone una sobrecarga en el sistema que perjudica el rendimiento obtenido, bien es cierto que dicha sobrecarga se puede reducir eligiendo el tipo de virtualización, en particular el uso contenedores. Esta penalización en el rendimiento es un aspecto a reducir así como el objetivo principal de este estudio.

Asimismo se ha comprobado que la configuración de ciertos aspectos del clúster no es trivial. Tanto o más relevante que el tipo de virtualización pueden ser otros parámetros, como son los relativos a la comunicación de los procesos dentro del clúster, la configuración de la topología de procesos o el método de difusión de resultados parciales entre dichos procesos.

En cuanto al uso de la virtualización, como resultado a destacar, se ha encontrado una excepción a la penalización previsible debido a la virtualización. Cuando se utiliza un clúster virtual que imita lo máximo posible a la infraestructura física se llegan a obtener rendimientos en torno al 110-120% a los obtenidos con el entorno nativo. Esta ventaja no se da siempre, solo se ha detectado cuando el problema a resolver no llegue a utilizar el 50% de la memoria total del clúster. Aunque sea en unos casos determinados, esto supone una ventaja significativa.

Palabras Clave: Computación de Alto Rendimiento, High Performance Computing, HPC, Supercomputación, Clustering, Virtualización, Contenedores, Containers, High Performance Linpack

Tabla de Contenidos

1 Introducción

Un campo de la informática con gran relevancia es el llamado HPC (*High Performance Computing* - Computación de Alto Rendimiento), el cual consiste en la máxima explotación de los recursos de un sistema informático compuesto por una o por varias máquinas con el fin de obtener una ventaja. Esta ventaja puede consistir en la resolución de grandes problemas inasumibles por una sola máquina, aumento en la velocidad de resolución de los mismos, alta disponibilidad de recursos, o combinación de varias de estas ventajas. Los métodos de introducirse en HPC son múltiples. Históricamente hablando, y es su escenario natural, la HPC está ligada al uso de grandes máquinas y sistemas de altas capacidades.

Pero HPC no es algo exclusivo de grandes sistemas, ya que con medios más modestos se pueden conseguir entornos HPC aceptables y suficientes para muchas situaciones que van del ámbito académico al empresarial. Desde hace unos veinte años va ganando en popularidad la creación de *clústers* por medio de ordenadores conectados a través de una red de comunicaciones utilizando componentes convencionales.

Esta clusterización por medio de máquinas convencionales es interesante desde varios puntos de vista, aunque predomina la disponibilidad y el factor económico. Otra ventaja importante que aporta la clusterización tipo *beowulf* es la facilidad en cuanto a la escalabilidad y/o sustitución de elementos que componen el clúster, tanto para actualizar componentes como para una reestructuración horizontal o vertical del clúster. Esta filosofía de acceso a la HPC es actualmente la predominante en escenarios de toda índole desde pequeños proyectos hasta los superordenadores más potentes del mundo como los que conforman el *Top500*[1].

La virtualización es una tecnología que cuenta con una larga trayectoria en grandes sistemas, como mainframes y servidores, con casi 50 años de historia, no obstante desde hace unos años se está consolidando en el mundo de los ordenadores personales y ámbitos más modestos gracias principalmente al soporte hardware en los microprocesadores de PCs (*Intel-VT* y *AMD-V*).

La virtualización permite desacoplar los recursos físicos del sistema de su sistema operativo nativo de forma que puede proporcionárselo a otros sistemas operativos. Esto proporciona ventajas más alla de la cohabitación de varios sistemas en una misma máquina, aunque sean consecuencia de ella: ahorro de espacio físico, mayor seguridad al haber mayor aislamiento entre procesos y servicios, mayor control y eficiencia de los recursos disponibles, facilidad de replicación de entornos, etc.

Ahora que se hacen más accesibles las tecnologías tanto relativas a la virtualización como a la HPC es el momento de probar hasta que punto puede ser tenido en cuenta la síntesis de ambos conceptos. Aunque es algo ya teorizado hace tiempo[1][3], el objetivo de este trabajo es el estudio de la sobrecarga y limitaciones que puedan suponer el uso de la virtualización basada en contenedores en un clúster destinado a HPC. Para realizar este estudio se midió el rendimiento bruto en distintos escenarios utilizando los mismos recursos HW totales, utilizando un entorno nativo y las distintas configuraciones y topologías de máquinas virtuales equivalentes.

Este documento está dividido de la siguiente forma: en la sección 2 se profundizará en las motivaciones de este estudio, haciendo hincapié en las distintas variaciones y opciones disponibles en cuanto a virtualización y clusterización; en la sección 3 se expondrá una situación y conceptos generales de las tecnologías a utilizar; en la sección 4 se discutirán las diferencias y semejanzas con otros trabajos relacionados con este estudio y en la sección 5 se expondrá el plan de pruebas diseñado para comprobar las hipótesis a verificar. En la sección 6 se discutirán los resultados obtenidos en las pruebas de la sección 5, en la sección 7 se expondrán las conclusiones obtenidas y por último en la sección 8 se hará mención al posible trabajo futuro derivado de este estudio.

[1] Lista de los 500 superordenadores más potentes del momento, con una actualización de 6 meses de los integrantes de dicha lista [`www.top500.org`].

2 Motivación y Objetivos

Como se ha explicado en la introducción, la virtualización es una tecnología que permite ciertas ventajas como consecuencia de desacoplar los recursos físicos de un sistema de su organización lógica. Esta abstración de los recursos lógicos de la infraestuctura física es bidireccional: se pueden tanto dividir y repartir dichos recursos en varios sistemas aislados entre sí, o bien se pueden unir varios recursos repartidos en varias infraestructuras para proporcionar la imagen de un único sistema de recursos centralizados.

No obstante, el uso de esta tecnología siempre tiene una contraprestación en cuestiones de rendimiento ya que el introducir la capa extra de abstracción tiene un coste computacional, lo que entra en conflicto con el el objetivo de la HPC.

Desde un punto de vista de HPC el rendimiento es uno de los factores que se busca maximizar ya que en HPC se intentan explotar al máximo los recursos de un sistema con el fin de obtener resultados de una forma más rápida y/o más precisa que no sería posible bajo un paradigma convencional de computación. Por eso HPC suele estar ligado con la computación paralela, donde se cuentan con varias unidades de procesamiento para poder aprovechar varias lineas de ejecución e intentar atajar un problema de mayor tamaño o en menor tiempo: esto es aplicable tanto a máquinas con multiples procesadores como en un sistema informático que utilice varias máquinas. En este último caso estaríamos hablando de un clúster y en ese caso habría que contar con otro componente imprescindible: la interconexión de los equipos.

Desde un punto de vista más cercano a este estudio, se puede mencionar el interés de esta combinación de virtualización y clusterización proporcionar uno o varios entornos aislados en los que al menos uno esté dedicado a tareas de computación de alto rendimiento. Esto puede ser necesario en escenarios donde haya gran trasiego de entornos, una mejor distribución y/o uso de los recursos disponibles, necesidad de servicios de alta disponibilidad o incluso, en un sentido más comercial, para poder saber que se puede esperar al ofrecer la infraestructura y/o la capacidad de cómputo como servicio (*IaaS – Infraestructure as a Service* – y *PaaS – Plaform as a Service* –, respectivamente).

El mayor escollo que puede haber al intentar unir la clusterización y virtualización en un ámbito HPC es la pérdida de rendimiento, siempre hablando en términos comparativos con un clúster *clásico* o *nativo*, debido a la sobrecarga de la virtualización ya que es el único elemento diferenciador entre ambos escenarios y el rendimiento es la principal característica de un entorno HPC.

Como se explicará a lo largo de este documento, para realizar los experimentos y comprobar sus características, se puso a punto un clúster[2] y se calcularon sus limitaciones, esto fue necesario para conocer el punto de partida y referencia de nuestros experimentos. Se diseñaron los distintos escenarios virtuales equivalentes con la misma infraestructura física que el clúster nativo y tanto dicho escenario nativo como los escenarios virtuales se sometieron a una serie de pruebas, tomando como referencia el límite del clúster nativo, con el fin de medir y comparar los distintos rendimientos obtenidos.

Como se verá más adelante, no hay un único escenario virtual equivalente al nativo debido a las posibles variaciones de distribución de recursos en un número distinto de máquinas virtuales. También la forma en la que se distribuye el problema importa, por lo que estas variaciones también deben ser tenidas en cuenta.

Todo esto se irá explicando a lo largo del documento pero podemos destacar que para conseguir el objetivo de medir y reducir las diferencias de rendimiento existentes entre un clúster nativo y uno virtual equivalente debemos hacer una selección de parámetros que pueden influir de manera significativa.

[2] Las características de las máqinas utilizadas se encuentran en la sección 5.1.

3 Panorámica general de las tecnologías

Tanto el mundo de la virtualización como el de clusterización son tecnologías que constan de múltiples conceptos y opciones diversas. A continuación se intentará proporcionar un contexto por separado de cada uno de estos ámbitos para que el lector pueda formarse una visión global del problema.

3.1 Virtualización

La virtualización es una técnica que permite abstraer, por medio de herramientas SW, el HW proporcionado por una máquina física (anfitriona) para simular el funcionamiento de otra máquina (huésped). Al conjunto de la máquina huésped y su SO propio se le llama máquina virtual, la cual utiliza los recursos de la máquina física anfitriona sin ser la realmente poseedora de los mismos. Esto implica ciertas ventajas potenciales como economía de recursos, ahorrro de espacio físico, coexistencia de multiples SO en una única máquina, mayor aislamiento entre procesos, etc.

No obstante, el uso de la virtualización también entraña ciertas desventajas, ya que una máquina virtual nunca tendrá el mismo rendimiento ni la calidad de acceso a todos los recursos del sistema de la misma forma que si estuviese ejecutándose de forma nativa en la máquina física. Esto es debido a que tanto la capa software que permite la virtualización como el sistema operativo nativo de la máquina física afectada supone otra capa de abstracción entre el HW y el SW final que requiere de capacidad de cómputo, espacio en memoria principal, y almacenamiento en disco. Aunque estos requerimientos sean mínimos, siempre van a existir y suponer un lastre.

Para poder realizar la virtualización es necesario por regla general lo que se conoce como un *hypervisor*. El *hypervisor*, si bien es un concepto bastante amplio, puede describirse como un interfaz entre el HW y el SW a virtualizar, ya que es la capa software que permite la virtualización y es la que servirá como enlace entre el HW real de la máquina física anfitriona y las máquinas virtuales huéspedes. Se pueden distinguir distintos tipos de virtualización dependiendo del papel que desempeñe el SO nativo (o mejor dicho, SO anfitrión) y la situación del *hypervisor* dentro de la jerarquía HW-SW.

Puede darse el caso de que el *hypervisor* necesite de un SO anfitrión ya insalado solo como un medio de poder instalarse en la máquina físicas. De forma que una vez instalado dicho *hypervisor*, éste es el que tiene la capacidad *directa* de gestión de los recursos HW de la máquina anfitriona. El *hypervisor* asigna los recursos a los sistemas operativos que tenga por encima, entre los cuales se incluye el SO anfitrión, por lo que en este caso se pierde un poco la perspectiva de SO anfitrión y los SO huésped virtualizados. Con este tipo de virtualización teóricamente se tiende a minimizar la sobrecarga producida tanto por el SO anfitrión como del *hypervisor*. Estos casos se conocen como virtualización de *Tipo I*, y entre sus ejemplos actuales se encuentra Xen.

Otro paradigma de virtualización se da si el *hypervisor* depende directamente del SO anfitrión. El SO anfitrión sigue siendo el que tiene prioridad de gestión de los recursos del sistema y el *hypervisor* es considerado un programa o servicio más del SO anfitrión. Dicho servicio se encarga de gestionar los distintos SO huésped (virtualizados). Si bien este tipo de virtualización supone una sobrecarga de cara al rendimiento en los entornos virtualizados, las ventajas radican en que se gana en flexibilidad y facilidad de administración y gestión de los entornos virtualizados. Este tipo de virtualización se conoce como virtualización como de *Tipo II*, y es la más conocida ya que tiene entre sus ejemplos a VirtualBox, VMWare o KVM.

Existe otra posibilidad que se escapa a la clasificación de virtualización de *Tipo I* o de *II*, que es la *Virtualización Basada en Contenedores*. Este tipo de virtualización no crea una máquina virtual completa, sino que proporciona espacios de nombre y usuario aislados entre

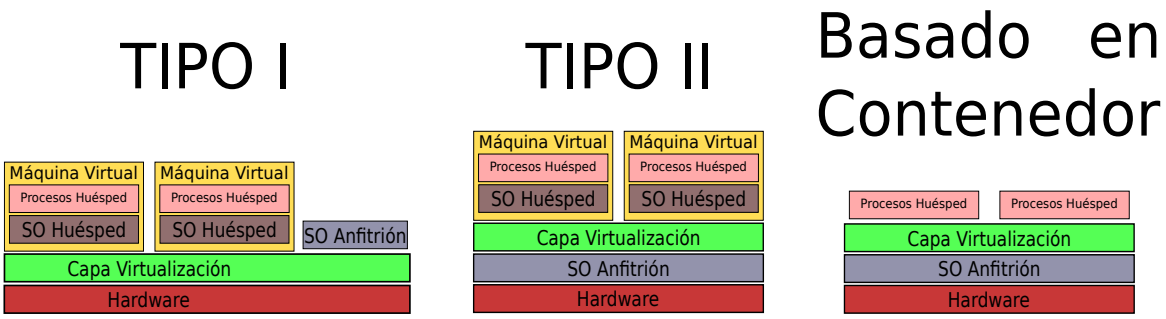

Fig. 1. Esquema jerárquico de elementos de HW y SW dependiendo del tipo de virtualización.

sí en los cuales se pueden lanzar procesos independientes y asignarles recursos físicos como si se tratasen de máquinas virtuales. Es por esto por lo que, aunque no sean máquinas virtuales completas, nos seguiremos refiriendo con este término a estos *contenedores* debido a que esto es transparente al usuario y sigue proporcionando la ilusión de que dichos espacios corresponden a distintias máquinas. Esto no se realiza por medio de un *hypervisor*, sino por medio del servicio *cgroups* del kernel Linux, por lo que existe una importante limitación que es que la imposibilidad de disponer de otro kernel asignado a dichos contenedores ya que utilizan el mismo kernel que el sistema anfitrión.

El uso de contenedores tiene tanto ventajas como desventajas: las ventajas incluyen un rendimiento excelente debido a una falta completa de sobrecarga y el hecho de que el núcleo tiene una visión global de todos los procesos que ejecutan en el sistema, por lo que la gestión de procesos puede ser más eficiente que si existieran dos núcleos independientes administrando conjuntos de tareas. La mayor de las desventajas es, obviamente, la imposibilidad de ejecutar un núcleo diferente en un contenedor, ya sea una versión diferente de Linux o directamente un sistema operativo distinto por lo que en el caso que ocupa este estudio, el mayor inconveniente es la falta de aislamiento entre procesos procedentes de la "máquina virtual" del contenedor y la máquina anfitriona[14]. Entre los ejemplos de este tipo de virtualización se encuentran *LxC - Linux Containers*, VServer, OpenVZ, o Docker. En este estudio, como veremos más adelante, se utilizaron contenedores basados en KVM.

3.2 Clusterización y HPC

La clusterización es una técnica ya conocida consistente en la comunicación de varias máquinas con capacidad de procesamiento de forma que puedan configurarse como un único sistema y así aumentar las capacidades de los recursos disponibles.

Esto es importante en varios aspectos especialmente ligados al uso de grandes sistemas informáticos y más concretamente en el ámbito de la HPC. Con la clusterización se incrementan las posibilidades de procesamiento en paralelo, las unidades de procesamiento disponibles y memoria principal del conjunto del sistema, lo cual normalmente está asociado a grandes equipos monolíticos. En comparación con una única gran máquina, la mayor diferencia radica en que por medio de la clusterización se permite una mayor flexibilidad tanto en el plano computacional, como económico, o estructural. Esto es debido a que, entre otras opciones y ventajas, un *clúster* se puede construir bajo demanda, se puede reformar/ampliar de una forma sencilla, o bien se puede adaptar a distintas necesidades de turno.

Históricamente existe un *benchmark* ligado a la medición de rendimientos de la CPU que es el High Performance Linpack (*HPL*), el cual es una evolución de *Linpack100* y *Linpack1000*. Al igual que sus predecesores, HPL trata de hallar la solución de un sistema lineal de ecuaciones representada por una matriz cuadrada de elementos con valores al azar.

Se diferencia de sus predecesores en dos aspectos. La primera de estas diferencias es el ámbito de actuación, ya que HPL fue especialmente pensado para sistemas con múltiples unidades de

procesamiento donde el paralelismo es uno de los puntos fuertes del sistema. El segundo aspecto en el que se diferencia HPL de sus predecesores es el tamaño del problema a resolver, ya que HPL no está limitado a un tamaño de matriz igual a 100 o 1000 elementos de lado, cosa que ocurre con Linpack100 y Linpack1000 respectivamente. Esto hace de HPL el *benchmark* ideal para nuestro propósito de darnos una primera idea de las capacidades de los recursos de los que disponemos.

No obstante, actualmente el *benchmark* HPL ha caido en desuso en el ámbito de la HPC precisamente por centrarse exclusivamente en el aspecto de rendimiento de CPU dejando de lado otras características y comportamientos relevantes en el mundo HPC tales como uso de red, lectura/escritura en disco, broadcasting, etc. Es por esto por lo que HPL esta siendo sustituido paulatinamente por el *benchmark* HPCC (*High Performance Computing Challenge*) el cual consta de 7 pruebas, entre las que se incluye HPL y otras 6 pruebas para suplir las carencias de HPL en cuestión de medición de aspectos relevantes de un sistema HPC.

Aún así este estudio se centrará en las diferencias de rendimiento de CPU en sistemas clúster virtualizados y nativos por medio de HPL, aunque se puedan extraer datos relativos al uso de red y comunicaciones.

3.3 Rocks Cluster

En el caso que ocupa este estudio, se realizarán las pruebas pertinentes por medio de la distrubición Rocks Cluster en su versión 6.1 y su actualización a la versión 6.1.1[3]. Rocks Cluster está basado en CentOS 6.5 y proporciona una serie de herramientas y facilidades ya preconfiguradas de creación y gestión de clústers tanto en su versión nativa como por medio de virtualización.

La virtualización que maneja Rocks se realiza a través del visor VMM (*Virtual Machine Manager*) por medio de máquinas virtuales KVM (*Kernel Virtual Machine*). Tanto las máquinas virtuales como las físicas tendrán que correr bajo Rocks Clusters en cierta configuración preestablecida[4] y aunque KVM es un *hypervisor* de Tipo II ya consolidado, las máquinas virtuales que se usan en Rocks están optimizadas utilizando el mismo kernel que las máquinas físicas que los alojan, por lo que son contenedores basados en KVM.

Aparte, en el entorno Rocks Clusters, se les llama contenedores a los nodos físicos donde se alojarán las máquinas virtuales que realizarán las tareas de nodos de computación, por lo que el termino *contenedor* puede ser confuso en este escenario.

[3] En el transcurso de desarrollo de este estudio se produjo esta actualización. Aunque no hay diferencias significativas en cuanto a uso o herramientas incluidas, si que se ha comprobado una mejora en la estabilidad de uso.

[4] Hay configuraciones de Rocks para *Frontend*, para nodos de computación, para máquinas físicas que alojarán máquinas virtuales, etc.

4 Trabajo Relacionado

En cuanto a trabajos de similar temática y objetivos se pueden encontrar *"An Updated Performance Comparison of Virtual Machines and Linux Containers"*[15] de Felter, Ferreira, Rajamony y Rubio; *"Performance evaluation of container-based virtualization for high performance computing environments"*[16] de Xavier, M. Neves, Rossi, Ferreto, Lange, y De Rose; y *"Performance Comparison of Hardware Virtualization Platforms"*[17] de Schlosser, Duelli y Goll. En estos estudios también se tratan experimentos basados en Linpack haciendo uso de clústers nativos y virtualizados bajo diferentes técnicas. Si bien ningún estudio de los nombrados hace exactamente los mismos experimentos ni utilizan los mismos entornos, si que podemos encontrar diferencias y semejanzas con todos ellos, las diferencias no son contradictorias con este estudio.

En el caso de [17]*"Performance Comparison of Hardware Virtualization Platforms"* de Schlosser, Duelli y Goll, pese a ser los experimentos de naturaleza distinta (no se hace uso de Linpack), se hallan resultados interesantes en cuanto al cuello de botella, a la penalización que se sufre con tamaños de mensaje pequeños y a la falta de aislamiento con KVM. Es díficil realizar una comparativa justa entre ambos trabajos, ya que en [17] se centran en la entrada y salida de bytes por paquetes en cada uno de los nodos (virtualizados o no) del cluster. No obstante, concluyen que se produce una pérdida importante de rendimiento en cuanto se utilizan paquetes de datos relativamente pequeños cosa que, como veremos, también se da en nuestro estudio. Otro aspecto interesante de [17] es que bajo la virtualización con KVM no se produce un aislamiento máximo entre los sistemas, fenómeno que sufrimos en este estudio al intentar rearrancar un cluster virtual (o bien al que se le ha caido uno de los nodos, o porque nosotros mismos lo habiamos apagado).

En el caso de [16]*"Performance evaluation of container-based virtualization for high performance computing environments"* de Xavier, M. Neves, Rossi, Ferreto, Lange, y De Rose, se utiliza una variacón HPCC. Esta versión de HPCC consta de varias pruebas también, entre las que se encuentra el uso de Linpack por lo que interesa la comparación con este estudio. No obstante, en dicho trabajo se limitan a realizar pruebas con una matriz de 3000x3000, lo cual se queda corto en relación a los tamaños se han probado en este trabajo. Prueban distintos entornos de virtualización, para lo cual obtienen resultados similares a los obtenidos por el entorno nativo, lo cual coincide con los resutados en nuestro trabajo para cargas y tamaños de problema pequeños. Este fenómeno no se da en problemas de tamaño relativamente grande, pero [16] no refleja este tipo de problemas, por lo que las comparaciones entre ambos trabajos se quedan ahí.

Y por último, si comparamos nuestro trabajo con [15]*"An Updated Performance Comparison of Virtual Machines and Linux Containers"* de Felter, Ferreira, Rajamony y Rubio, el cual parece más interesante, también se hace uso de Linpack dentro de una serie de pruebas de otro tipo de versión de HPCC. Entre dichas pruebas destacan la comparación de un entorno de virtualización basado en contenedores (Docker) y la plataforma de virtualización KVM comparadas con un escenario nativo. En su caso, la plataforma KVM da muy malos resultados (algo menos del 50% del rendimiento obtenido por el entorno nativo) y la virtualización basada en contenedores muy buenos resultados (similares al nativo) por lo que, en conjunto, parece coincidir en ciertos aspectos con los resultados obtenidos en nuestro trabajo, el cual recordemos utiliza unos contenedores basados en KVM. Este estudio no detalla el uso de memoria ni tamaño de problema, por lo que es difícil hacer una comparación exhaustiva.

5 Pruebas Experimentales

A lo largo de esta sección explicamos el entorno experimental. para lo cual describimos la infraestructura HW y el SW utilizado. En cuanto al aspecto SW, se hará especial énfasis en ciertos detalles clave del *benchmark* HPL para mejor comprensión de los experimentos realizados. Una vez descrito el entorno experimental y explicado los conceptos clave definimos el banco de pruebas y explicamos de forma pormenorizada cada una de las pruebas experimentales.

5.1 Descripción del entorno experimental

– **Entorno HW**
Todas las pruebas se realizaron en un entorno compuesto por máquinas físicas con microprocesador Intel i7 3770 a 3'9GHz, con 4 cores físicos cada uno y con *HyperThreading* activado por defecto[5], es decir, 8 cores lógicos por cada máquina. Asimismo, cada máquina física cuenta con 8 GB de RAM.
Todas las máquinas físicas utilizadas en este estudio se corresponden con esta configuración independientemente de su función dentro del clúster ya sea como *frontend*, nodo de computación o contenedor de máquinas virtuales.

– **Entorno SW**
Todas las máquinas tienen instalado Rocks Cluster 6.1.1 como sistema operativo el cual está basado en CentOS 6.5. Esta distribución ya trae las herramientas necesarias para formar un clúster y/o hacer uso de virtualización, como el gestor de tareas *Sun GridEngine*, KVM y el gestor de máquinas virtuales como el SW más relevante de nuestros propósitos.

En cuanto a la configuracion de escenarios que hagan uso de máquinas virtuales y la distribución de las mismas, tal y como se muestra en la figura 2, al disponer de máquinas físicas con 8 unidades de procesamiento se pueden utilizar 4 distribuciones distintas de unidades de procesamiento por máquina virtual para formar clústers lo más homogéneos posibles: podemos crear una única máquina virtual por máquina física que se adueñe de todos los recursos de la misma[6], podemos maximizar el número de máquinas virtuales al crear una por cada unidad de procesamiento de la máquina física y, podemos hacer dos configuraciones *intermedias* a esos dos extremos, podemos hacer 2 máquinas virtuales por máquina física cada una con 4 unidades de procesamiento o bien podemos hacer 4 máquinas virtuales por máquina físca cada una con 2 unidades de procesamiento.

En cuanto al *software* utilizado para realizar los experimentos, el *benchmark* escogido fue *High Performance Linpack* (HPL). HPL trata de resolver un sistema lineal de ecuaciones el cual está representado por una matriz distribuida entre todos los ordenadores que componen el clúster[7]. Los cuatro parámetros más importantes a definir en HPL son el tamaño del problema a resolver (N), la granularidad (NB), la topología de distribución de procesos (PxQ) y algorítmos de difusión de panel. Así mismo, HPL precisa de una librería matemática que le proporcione las funciones matemáticas apropiadas, como veremos más adelante.

- **N y *Problema de Tamaño Estándar***

  El tamaño del problema (N) es sin duda el parámetro más importante en un test realizado con HPL. N indica el lado de la matriz a resolver.

[5] *HyperThreading* duplica el número de cores físicos que verá el SO
[6] Réplica virtual.
[7] En el Anexo A se profundiza acerca de la historia, características y parámetros de HPL.

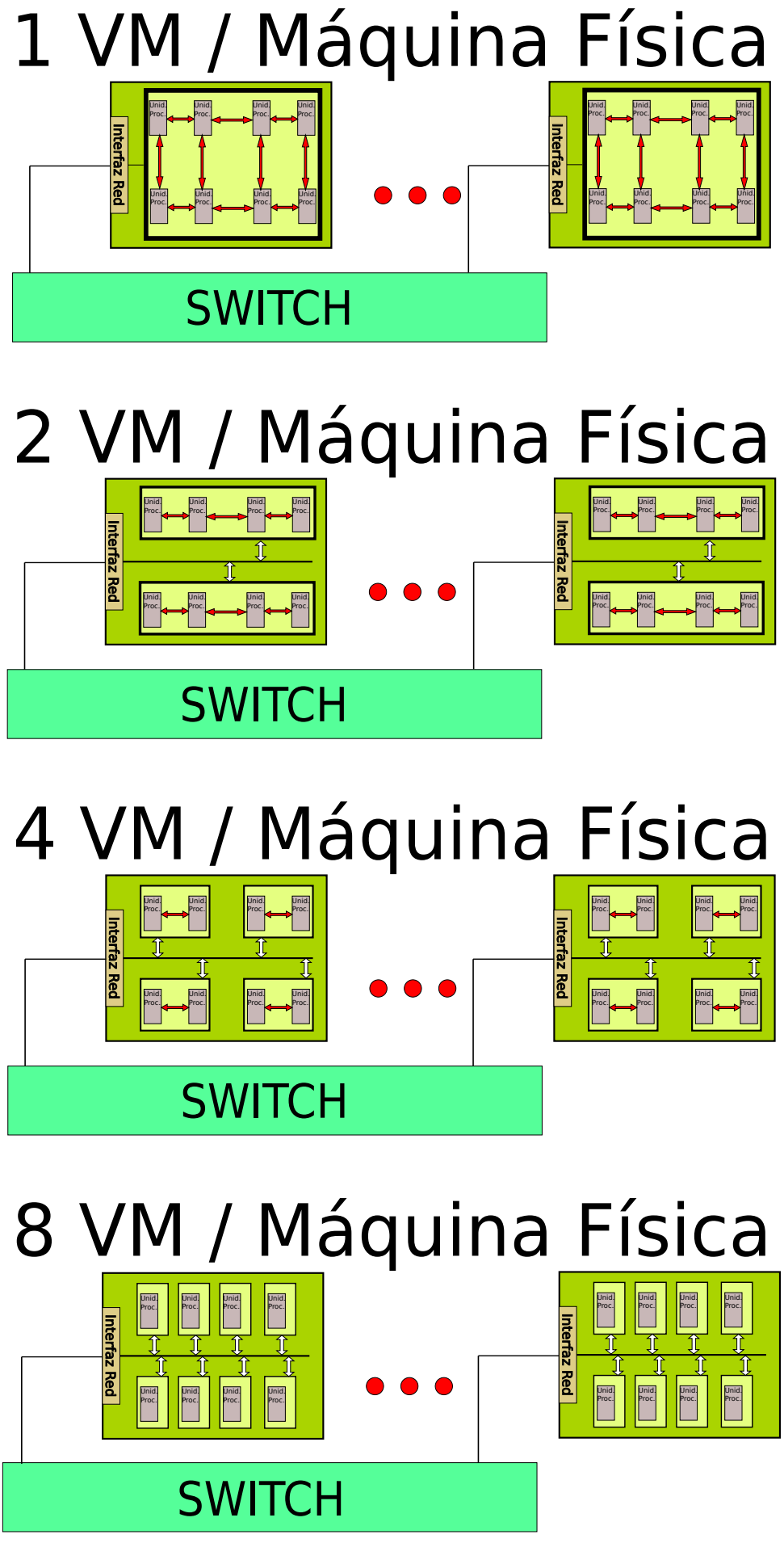

Fig. 2. Esquema que muestra las distintas distribuciones de las máquinas virtuales en un contenedor y las comunicaciones existentes. Las flechas rojas representan las comunicaciones internas en una misma VM, y las flechas azules representan las comunicaciones internas dentro de un contenedor (pero externas a las VM). Las comunicaciones existentes entre distintos contenedores/Máquinas Físicas no se muestran, pero se da por dado que existen y son a traves del *switch*. Nota: Por simplificar en el gráfico, en el caso de las comunicaciones internas dentro de una VM (flecha roja) no se muestran todas, solo unas cuantas para ejemplificar que existen.

Un concepto a tener en cuenta de cara a las pruebas es el término que utilizamos en este trabajo llamado *Problema de Tamaño Estándar*, ya que es el principal elemento en muchas de las pruebas. En este documento se hace referencia a un *Problema de Tamaño Estándar* a un problema HPL con N=10000. Esto es, la resolución de una matriz cuadrada con 10000 elementos de lado que representa un sistema lineal de ecuaciones.

Se toma este tamaño de problema como estándar por varios motivos. Parece una evolución lógica en cuanto al tamaño de problema incrementar en un orden de magnitud el tamaño del problema respecto a los *benchmark* Linpack100 y Linpack1000, los cuales recordemos que son los precedentes de HPL, no orientados a paralelismo y limitados a N igual a 100 y 1000 respectivamente.

Como segundo motivo, el modelo de N = 10000 se adapta perfectamente a nuestro entorno experimental. Una matriz de ese tamaño 10000 ocupa aproximadamente unos 750MB en memoria principal, lo cual es representa un 75% de memoria en máquinas que tengan 1GB de memoria. Esto se ajusta en ciertos experimentos contemplados en el banco de pruebas, en los que el escenario está compuesto por máquinas con una unidad de procesamiento y 1GB de memoria asignada[8].

- **NB y Granularidad del Problema**
  La granularidad del problema se puede ajustar a través del *tamaño de bloque* (NB). NB se indica en número de elementos de la matriz a intercambiar entre procesos por cada mensaje empleado. Esto predispone las comunicaciones empleadas a la hora de resolver el problema, ya que a menor tamaño de mensaje, mayor granularidad y mayor número de mensajes necesarios para comunicar los sucesivos cambios que se vayan produciendo en la matriz. Y viceversa, a mayor tamaño de mensaje es necesario menor numero de comunicaciones.
- **PxQ y Topología de distribución de procesos**
  En HPL, la matriz a resolver se divide en distintas porciones cada una de las cuales estará asociada a un único proceso. Cada una de estas porciones se llama *panel*.

  El número de paneles a utilizar se define tanto en las características de HPL como en los parámetros de ejecución de cualquier tarea paralela en el sistema operativo. Es importante que coincida el número de paneles definidos en HPL con el número de tareas paralelas a lanzar por el sistema operativo. Si el número de paneles definidos en HPL supera al número de tareas paralelas asignadas en el SO para la ejecución, entonces HPL nunca será ejecutado debido a que necesita más unidades de procesamiento que se han solicitado al SO. Por el contrario, si el número de paneles definidos en HPL no alcanza el número de tareas que se espera el sistema operativo, entonces HPL se ejecutará pero sin utilizar todas las unidades de procesamiento reservadas para la tarea.

  En HPL, el número de paneles a utilizar para resolver un problema viene dado por los parámetros P y Q que indican en cuantas filas y columnas queremos subdividir la matriz a resolver. Si multiplicamos P y Q, obtendremos el número final de paneles. Pero no solo el número paneles del problema/unidades de proceso a utilizar es importante: tan importante como utilizar el máximo número de unidades de proceso disponibles puede ser la forma en que se alcanza ese máximo número de unidades de proceso por medio de la multiplicación PxQ, ya que de esta forma se predisponen las comunicaciones entre procesos.

  Es decir (y poniendo un ejemplo real, tal y como se muestra en la figura 3), utilizando una rejilla de procesos PxQ igual a 1x16 o utilizando una rejilla de procesos PxQ igual a 4x4, la matriz a resolver se divide en 16 paneles que se asignaran a 16 procesos distintos,

[8] Ideal para este tipo de estudios en los que se recomienda utilizar entre un 70 y 80% de la memoria en cada máquina del clúster.

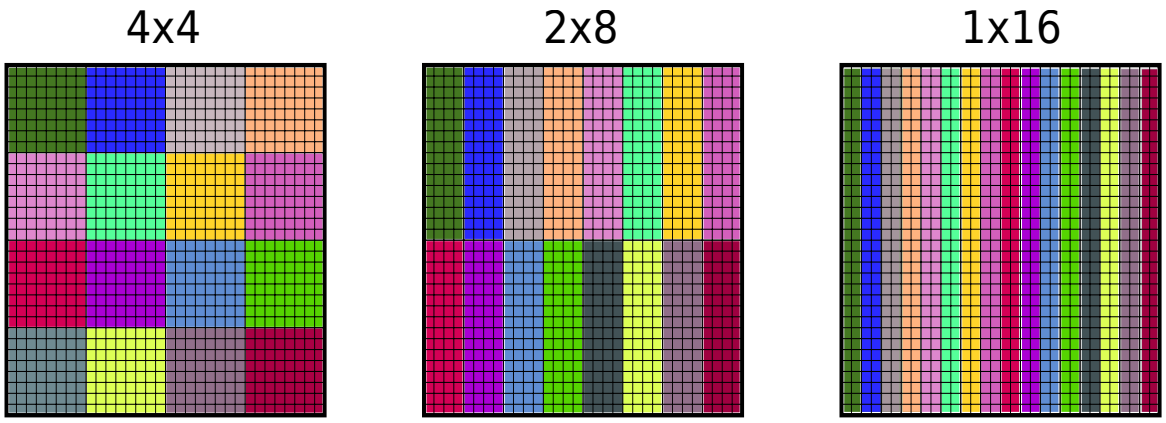

Fig. 3. Ejemplo de distribución de un problema utilizando 16 procesos. La figura de la izquierda se corresponde con una rejilla de procesos 4x4 (compensada), la del centro con una rejilla 2x8 (algo de la derecha) y la de la derecha con una rejilla 1x16 (muy descompensada). Cada proceso solo comunica sus resultados parciales con los procesos colindantes, por lo que el uso de una rejilla compensada implica una mayor carga de la comunicación entre procesos (pero mayor fluidez en la comunicación), y una rejilla más descompensada implica menor carga relativa de las comunicaciones pero la comunicación de resultados parciales es menos fluida.

cada uno de los cuales utilizará una única unidad de procesmiento. La diferencia radica en que forma se comunicarán dichos procesos:
Utilizando una rejilla 1x16 las comunicaciones que se producen son *punto a punto* (o *en línea*) de forma que cada uno de los procesos solo se comunica con otros 2 procesos. Es a lo que se llamará una distribución, topología o rejilla de procesos más descompensada.
Utilizando una rejilla 4x4, la matriz se dividirá y distribuirá también entre 16 procesos, pero estos no se comunican entre sí de una forma punto a punto, sino que dichas comunicaciones se extenderán en un "plano de procesos" y tenderá a haber mayor número de comunicaciones totales. Esto tiene la ventaja de que hace más fluidas las comunicaciones entre procesos y es por lo que se dice que esta topología o rejilla de procesos es más compensada.
De forma similar, si utilizamos 32 procesos tal y como ocurre en muchas de las pruebas de este estudio, 1x32 sería la distribución de la rejilla de procesos PxQ más descompensada posible, y 4x8 la distribución más compensada.

- **Difusión del Panel**
  La difusión del panel (o *panel broadcast*) en HPL es el algoritmo que se aplica para coordinar las comunicaciones entre procesos. Una vez que un proceso resuelve parte del panel asignado, este proceso debe comunicar dicha actualización a los demás procesos. Dicha comunicación siempre se hará entre los procesos que manejen paneles colindantes.
  Aunque HPL ofrece 6 algoritmos distintos para difundir paneles, realmente son dos algoritmos de tipo *Long*, uno de tipo *Ring*, y una variación de cada uno de estos tres algoritmos nombrados. La diferencia radica en que la difusión tipo *Ring* es asíncrona y el proceso que genera un cambio en los datos del panel comunica inmediatamente dicho cambio a los procesos colindantes pertinentes.
  Dentro de los algoritmos tipo *Ring* que ofrece HPL hay dos grandes variantes: una utiliza forma de comunicación más lineal entre procesos colindantes, mientras que otra utiliza saltos y tiende a comunicarse tanto con los procesos más cercanos como con los más lejanos. En este estudio se utilizó esta ultima variante.
  Los algoritmos tipo *Long* sin embargo exigen una sincronización entre todos los procesos de forma que a partir de un mismo instante, los mensajes de cada proceso son divididos a su vez en Q porciones iguales que serán compartidos en cada paso a otros dos procesos distintos. Al cabo de Q-1 pasos todos los procesos habrán compartido y recibido todas las actualizaciones pertinentes entre sí. Recordemos que Q es el valor que se corresponde

con lo indicado en PxQ en el archivo de configuración de ejecución de HPL.

– **Banco de pruebas**
El banco de pruebas diseñado para este trabajo está dividido en 3 fases, cada una de las cuales se compone de una o varias pruebas tal y como se desgrana en la tabla 1.

La primera fase consiste en una serie de pruebas de configuración (Cx) con el fin de comprobar cual es la configuracion idónea del clúster[9]. En esta fase se comprueban varios componentes y atributos del clúster con el fin de esclarecer que opción es la mejor en cada caso y así obtener el mejor clúster posible de cara al resto de pruebas.

Código de la prueba	Entorno	Objetivo
Cx	Físico	Configuración del entorno experimental. Todas las pruebas Cx se corresponden a pruebas de configuración de distintos requisitos o aspectos del clúster (interconexión, bibliotecas, ...) y tratan simplemente de establecer un punto de partida para las siguientes pruebas, y/o explicar porque se eligió una alternativa en detrimento de otra.
R1	Físico	Comprobar los límites del clúster en su configuración física con el fin de establecer un marco de referencia de cara a futuras pruebas que intervengan configuraciones virtuales del clúster.
V1	Físico + Virtual	Comprobar mediante un problema de tamaño estándar las diferencias entre un clúster físico y uno virtual equivalente.
V2	Físico + Virtual	Comprobar mediante problemas de tamaño variable (desde uno de tamaño estándar hasta uno de tamaño equiparable al total de la memoria disponible en el clúster) las diferencias entre un clúster físico y uno virtual equivalente.
V3	Físico + Virtual	Comprobar mediante un problema de tamaño estándar y otro de tamaño equiparable a la memoria disponible en una máquina, las diferencias entre un clúster virtual concentrado en una única máquina y un clúster virtual distribuido en varias máquinas.
V4	Físico + Virtual	Comprobar mediante un problema que represente una carga constante en memoria el comportamiento en un clúster (tanto nativo como virtual, distribuido y concentrado) compuesto por un mismo número de máquinas.

Tabla 1. Tabla de plan de pruebas

Una vez ya definidas y configuradas todas las características esenciales del clúster se procede a realizar la segunda fase del banco de pruebas. Esta segunda fase se compone de una única prueba en la que se comprueban los límites de capacidad física del clúster[10] (R1). Esto consistió en intentar resolver el problema de mayor tamaño posible sin utilizar ningún tipo de virtualización, es decir manteniendo un entorno puramente físico. Así se tuvo un marco de referencia de cara a comparar el impacto de la virtualización.

Una vez ya configurado el clúster y comprobado sus capacidades máximas se procedió a realizar la tercera fase, elemento central de este estudio [11]. La tercera fase del banco de pruebas consistió en comparar un entorno completamente nativo respecto a sus escenarios virtuales equivalentes al intentar resolver problemas de distinto tamaño y granularidad. El

[9] Veáse la sección 5.2, Configuración: Interconexión de red, alojamiento de procesos, selección de bibliotecas y método difusión de panel HPL.
[10] Veáse la sección 5.3, Prueba en Clúster Real 1 (R1): Capacidad del Entorno Nativo.
[11] Veáse la sección 5.4, Pruebas Entorno Físico Vs. Virtual y subsecciones correspondientes.

tamaño de los problemas variaron desde un *tamaño estándar* (V1), hasta problemas que por tamaño se acerquen a los límites del clúster en su configuración nativa, ya comprobados en la prueba R1 (V2).También se hicieron pruebas respecto a como influye la distribución de las máquinas virtuales dentro de la infraestructura HW dada (V3) y como se comporta un clúster que se escala horizontalmente bajo una carga de trabajo siempre proporcional a la memoria total del clúster (V4).

5.2 Configuración: Interconexión de red, alojamiento de procesos, selección de bibliotecas y método difusión de panel HPL.

A la hora de comenzar los experimentos se tiene como situación de partida un clúster sin una configuración concreta en ciertos atributos clave, tanto HW como SW: el uso o no de *HyperThreading*, el tipo de interconexión física a utilizar, la forma en la que el gestor de tareas del clúster asigna los procesos en las máquinas, la biblioteca matemática a utilizar por el *benchmark* HPL y el algoritmo de difusión de los paneles a utilizar en HPL. En el caso de utilizar otro *benchmark* o un clúster con características distintas, los atributos clave a ajustar pueden variar.

Se realizaron una serie de pruebas en espiral en las que se comprobó el comportamiento de un clúster físico variando el valor de un atributo en concreto con el fin de esclarecer que opción es la mejor para cada uno de esos atributos clave y así contar con una configuración defnitiva y estable del clúster en el resto de pruebas.

El clúster a configurar está compuesto por 5 máquinas similares a las ya descritas al inicio de la sección 5.1: una de ellas actuó como *Frontend* y las otras 4 como nodos de computación.

Todas las pruebas de este apartado se realizaron ejecutando un *Problema de Tamaño Estándar)*[12], con un tamaño de bloque NB variable en la franja de 32-256 elementos por mensaje.

La configuración por defecto consistieron en utilizar la difusión bloques por el método *Increasing-ring (modificado)*, la biblioteca algebraica GotoBLAS2, un sistema de interconexión *Ethernet Gigabit*, y los procesos se distribuirán por *Round Robin*. No obstante, esta fue la configuración por defecto y se realizaron pruebas variando alguno de estos atributos para comparar alternativas y quedarnos con las mejores opciones posibles, tal y como se explica a continuación:

Partiendo de un punto de vista más cercano al HW, y finalizando con un punto de vista más SW, se tienen los siguientes atributos clave del clúster y que opciones se probaron para tal atributo:

- **Inerconexión de Red**: *Ethernet 10/100* o *Ethernet Gigabit*. Se realizó esta prueba por tener la posibilidad de comprobar el comportamiento del clúster bajo ambos elementos estructurales. Se puede predecir claramente que mediante el uso de Ethernet Gigabit el rendimiento del clúster será superior al obtenido mediante un clúster similar que use Ethernet 10/100 como interconexión interna. Se probaron ambas interconexiones con el fin de comprobar la diferencia de rendimiento entre ambas, identificar posibles comportamientos a tener en cuenta y obtener experiencia de cara a entornos que sólo cuenten con la posibilidad de utilizar Ethernet 10/100.
- **Alojamiento de procesos (SGE)**: *Fill Up* o *Round Robin*. El gestor de tareas es el componente central de un clúster (virtualizado o no) a nivel SW, ya que con el se asignan las tareas entre los elementos del clúster. El gestor de tareas utilizado por Rocks Cluster es SGE (*Sun Grid Engine*), y puede repartir los procesos entre los nodos de dos formas distintas: *Fill Up* o *Round Robin*. Mediante *Fill Up*, SGE empieza a asignar procesos a una máquina del clúster hasta que dicha máquina no tiene más unidades de procesamiento disponibles,

[12] N=10000.

momento en el cual empiezan a asignarse procesos a otra máquina hasta que de nuevo no tenga unidades de procesamiento disponibles, y así sucesivamente hasta que no queden procesos a repartir. Mediante *Round Robin* los procesos se van repartiendo equitativamente entre las máquinas que componen el clúster, de forma similar a como se reparten las cartas a los jugadores en una partida. No hay información acerca de que método se aconseja, con cual se obtiene mejor rendimiento, o preferencias de ningún tipo. Como hay que decidir un método, se optó por probar ambos métodos y comparar resultados.
- **Bibliotecas de Álgebra Lineal**: *GotoBLAS2* u *OpenBLAS*. HPL precisa de una biblioteca de funciones matemáticas para poder desempeñar su labor. La biblioteca recomendada por la documentación oficial de HPL es GotoBLAS2 el cual es un proyecto abandonado pero continuado por el proyecto OpenBLAS. La documentación oficial no recoge información al respecto y es fácil presuponer que OpenBLAS puede suponer una ventaja debido a la evolución que significa respecto a GotoBLAS2. Ante la duda respecto al comportamiento de ambas bibliotecas, se probaron las dos y comparar resultados.

- **Difusión del Panel (HPL)**: *Increasing-ring (modificado)* o *Long (modificado)*. En la sección 5.1 se habló de la difusión de paneles en HPL. La documentación indica que puede ser más rápida una difusión tipo *Long* que una tipo *Ring* en escenarios en los que la capacidad de cómputo sea, proporcionalmente, más rápida que la capacidad de interconexión de la red interna[13]. Ante tal recomendación se decidió probar ambos métodos pero en sus versiones modificadas ya que estas sí representan una mejora respecto a su versión original.

Hubo un atributo clave que no se configuró en esta serie de pruebas que fue el uso de *HyperThreading*. Al tener este atributo gran influencia sobre el rendimiento se pospuso la elección de su uso a la espera de tener el resto de atributos configurados definitivamente. No obstante, el uso o no de *HyperThreading* se verá en la prueba referente a las capacidades físicas del clúster (Prueba R1).

De forma que se realizaron 4 pequeñas pruebas para cada una de las cuales se resuelve un *Problema de Tamaño Estándar* de dos formas respecto a la configuración por defecto antes mencionada, una por cada opción a tener en cuenta. Y, exceptuando el atributo de interconexión de red interna, no se pudo realizar una previsión rotunda acerca de que opción resulta mejor para cada atributo clave. En base a la información disponible, puede resultar mejor utilizar OpenBlas en vez de GotoBLAS2 y puede resultar mejor utilizar una difusión de los paneles en HPL con un método tipo *Long* antes que una tipo *Ring*. No obstante, no dejan de ser hipótesis que no se resolvieron hasta que no se probaron las distintas opciones ya mencionadas.

Finalizado a este punto, ya se tuvo el clúster configurado de forma definitiva, resultando que es mejor no variar la configuración predeterminada (al menos en el escenario probado), de tal forma que la configuración del clúster fue:

- **Inerconexión de Red**: *Ethernet Gigabit*.
- **Alojamiento de procesos (SGE)**: *Fill Up*.
- **Bibliotecas de Álgebra Lineal**: *GotoBLAS2*.
- **Difusión del Panel (HPL)**: *Increasing-ring (modificado)*.

No obstante esto es un adelanto de los resultados obtenidos tal y como se detallará al inicio de la sección 6.

[13] Como puede ser este caso. Téngase en cuenta que el sistema de interonexión habitual en HPC suele ser más rápido, como Infiniband.

5.3 Prueba en Clúster Real 1 (R1): Capacidad del Entorno Nativo

El objetivo de esta prueba fue doble: comprobar el límite que admite el clúster en cuanto al tamaño del problema que es capaz de resolver, y utilizar esta prueba para comprobar el impacto del uso de *HyperThreading* en el rendimiento de un entorno HPC nativo, es decir, sin utilizar máquinas virtuales.

Por lo que para esta prueba se crearon dos clústers con las siguientes características:

- Cada uno de los clústers utilizó una máquina física haciendo de frontend y 4 máquinas físicas haciendo de nodos de computación.
- Uno de los clústers tuvo *HT* activado y el otro desactivado.

En ambos casos la interconexión interna utilizada fue Ethernet Gigabit[14] y se utilizó *Fill Up* para distribuir los procesos de HPL, el cual se ejecutó por medio de la biblioteca GotoBLAS2 tal y como se explicó en la seción 5.2.

El número de procesos a lanzar varió entre los dos clústers. Esto es debido a que *HyperThreading* hace que el sistema operativo tenga la ilusión de que cada máquina física tenga el doble de unidades de procesamiento de las que realmente tiene. Esto es posible replicando gran parte de los registros de la CPU, así cada máquina del clúster puede, en teoría, lanzar el doble de procesos simultáneos de los que normalmente podría ejecutar.

Por la naturaleza del experimento, debido a la variación de unidades de proceso entre clústers por el uso de *HyperThreading*, la rejilla de procesos PxQ varió de un clúster a otro. En cualquier caso se probaron todas las posibilidades de rejilla PxQ realizables en cada caso tal y como se muestran en la tabla 2.

P	Q
1	16
2	8
4	4

P	Q
1	32
2	16
4	8

Tabla 2. A la izquierda, tabla de rejillas a probar cuando el *HyperThreading* está **desactivado**. A la derecha, tabla de rejillas a probar cuando el *HyperThreading* está **activado**. El resultado de multiplicar P por Q da el número de procesos a utilizar por el clúster en cada caso.

En cuanto a la prueba en sí, implica que al utilizar 4 máquinas como nodo de computación en cada uno de los clústers en el que está *HT* desactivado se tiene un clúster de 16 cores, mientras que en el clúster que tiene *HT* activado se cuenta con 32 cores[15] pese a utilizar también 4 máquinas físicas como nodos de computación.

Se entiende que una rejilla de procesos PxQ es más compensada si los coeficientes P y Q tienden a ser iguales y, por el contrario, una rejilla PxQ es más descompensada si P y Q son los números más desiguales posibles. En la prueba que nos ocupa, para el caso de tener *HyperThreading* activado la rejilla PxQ igual a 4x8 es la más compensada y 1x32 la más descompensada. Si *HyperThreading* está desactivado, la configuración de rejilla de procesos 4x4 es la más compensada y 1x16 la más descompensada. Recuérdese que según la documentación oficial de HPL, a la hora de configurar la rejilla de procesos, P debe ser igual o menor que Q.

Para cada uno de los clústers se probaron tamaños de bloque NB del rango entre 32 y 256 elementos en problemas de tamaño N igual a 10000, 20000, 40000, 50000 y 60000.

Tal y como se muestra en la tabla 3, se fijó N igual a 10000 como límite inferior de tamaño de problema por ser el problema estándar ya probado en Cx. Así mismo se fijó como límite

[14] Obviamente, no se realizó la prueba simultaneamente en ambos clústers pese a utilzar el mismo switch.
[15] ¡Atención! No son cores físicos.

N	%Mem
10000	15.5
20000	31.0
40000	61.9
50000	77.4
60000	92.9

Tabla 3. Los tamaños de matriz probados asociados al porcentaje de memoria que estos representan en el conjunto de nodos de computación utilizado.

superior de orden de problema 60000 por considerarse una cifra redonda lo suficientemente grande y cercano al límite teórico de la memoria. Un problema HPL con N igual a 60000 ocuparía aproximadamente un 93% de la memoria principal del clúster. Se esperaba que el *benchmark* HPL no se realizase con éxito para alguna prueba entorno a valores altos de N debido a que representaba un porcentaje demasiado alto de la memoria total del clúster, como el ya mencionado caso de N igual a 60000. Ante estos casos, es de esperar que bien falle algún nodo de computación del clúster o bien que HPL no sea capaz de terminar su ejecución. El coste computacional de HPL en relación con el tamaño S del problema a resolver es de orden cúbico[8], es decir R = O(S^3), siendo S el tamaño del problema y el R rendimiento [16]. A su vez existe una relación inversamente proporcional entre rendimiento y tiempo empleado en resolver un problema $R \sim 1/T$ y una equivalencia $S(N) = N^2$.

Si se toma el rendimiento obtenido por el problema con N = 10000[17] como unidad de referencia del rendimiento, R_{10K}, el rendimiento debería escalar en una proporción algo inferior a la raiz cúbica de la proporción del tamaño del problema en relación al *Problema de Tamaño Estándar*.

$$\Delta R = \frac{R(N)}{R_{10K}} \sim \sqrt[3]{\frac{S(N)}{S(10K)}} = \sqrt[3]{\frac{N^2}{(10K)^2}}$$

Toda esta relación entre lado de la matriz a resolver, tamaño de la misma en elementos, proporción que guarda ese tamaño de problema en relación con un *Problema de Tamaño Estándar*, y previsión de rendimiento, están recogidas en la tabla 4.

N	S	ΔS	Previsión ΔR
10K	100M	1	R_{10k}
20K	400M	4	1.6*R_{10k}
40K	1600M	16	2.5*R_{10k}
50K	2500M	25	2.9*R_{10k}
60K	3600M	36	3.3*R_{10k}

Tabla 4. Relación de proporciones en tamaño y rendimiento previsto para los tamaños de problema a probar en el experimento R1. Para la prueba V2 se sustituye N=60K por N=50800, el cual tiene valores muy similares a N=50K. En cualquier caso N es el orden de la matriz a resolver, S el tamaño total de elementos de la matriz (cada uno es un *float* de doble precisión, es decir, que ocupa 8Bytes cada uno), ΔS es la relación del tamaño de problema respecto al problema de orden N = 10K, y Previsión ΔR es la predicción de rendimiento de un problema (en función del rendimiento dado por el problema de orden 10K).

[16] Se usa S (*size*) para evitar una posible confusión con el tamaño del lado de la matriz en elementos N, o con tiempo T.
[17] *Problema de Tamaño Estándar.*

5.4 Pruebas Entorno Físico Vs. Virtual

5.4.1 Prueba en Clúster Virtual 1 (V1): Tamaño Estándar

Esta prueba tuvo por objetivo una primera comparación de un clúster físico y uno virtual. Para ello se crearon distintos clústers, siempre compuestos por 4 máquinas físicas como nodos de computación o contenedores de máquinas virtuales según fuese el caso. Los escenarios a probar fueron:

- Un clúster físico compuesto por 4 máquinas físicas como nodos de computación con HT activado.
- Un clúster virtual compuesto por 4 nodos de computación virtuales, cada máquina virtual maneja 7960 MB de memoria y 8 cores lógicos. Cada máquina física aloja 1 máquina virtual.
- Un clúster virtual compuesto por 8 nodos de computación virtuales, cada máquina virtual maneja 3980 MB de memoria y 4 cores lógicos. Cada máquina física aloja 2 máquinas virtuales.
- Un clúster virtual compuesto por 16 nodos de computación virtuales, cada máquina virtual maneja 1990 MB de memoria y 2 cores lógicos. Cada máquina física aloja 4 máquinas virtuales.
- Un clúster virtual compuesto por 32 nodos de computación virtuales, cada máquina virtual maneja 995 MB de memoria y 1 core lógico. Cada máquina física aloja 8 máquinas virtuales.

Es decir, se realizarón la ejecución del benchmark todas las posibles configuraciones utilizando 4 máquinas físicas con *HyperThreading* activado como nodos de computación o contenedores de máquinas virtuales, tal y como muestran en la tabla 5 (izquierda):

Físico/Virtual	Núm. Máqinas	LogCore/Máquina
Físico	4	8
Virtual	4	8
Virtual	8	4
Virtual	16	2
Virtual	32	1

P	Q
1	32
2	16
4	8

Tabla 5. A la izquierda, tabla de configuraciones probadas en las pruebas V1 y V2. A la derecha, tabla de rejillas de procesos a probar considerando que el *HyperThreading* está **activado**.

En todos los escenarios se ejecutó el benchmark HPL con un *Problema de Tamaño Estándar* (N = 10000). Se ajustaron los tamaños de bloque (NB) entre 32 y 256 elementos con el fin de averiguar que granularidad es la adecuada para resolver el problema. Dichos factores N y NB son de lejos los más relevantes en la ejecución de HPL pero como todos los escenarios ejecutan los mismos N y NB estos factores no aportan información relevante desde un punto de vista de uso de virtualización. Por este motivo nos centramos en las diferencias entre configuraciones del clúster, y la configuración de las rejillas PxQ, ya que estas diferencias acentuadas por la virtualización sí que pueden suponer una ventaja (o desventaja).

En cuanto al tipo de configuración, se cree que la configuración nativa[18] debería ser la que mejor resultados obtuviese debido a la carga extra y ligera reducción en el acceso a los recursos comunes en entornos virtuales. Partiendo de esa base, a medida que hubo más comunicaciones en general peor debió ir el rendimiento. De forma que se previó que la configuración virtual con 32 máquinas será la que peor resultados obtendría ya que dicho clúster tiene el mayor número de máquinas a coordinar, por lo tanto mayor comunicación entre procesos, y además cada una de las máquinas virtuales implicadas cuenta con la menor capacidad de cómputo de todos los escenarios a probar.

[18] O real, o física, como quiera decirse.

5.4.2 Prueba en Clúster Virtual 2 (V2): Tamaños Grandes

En esta prueba se probó con distintos tamaños de problema HPL utilizando tanto un clúster físico en su forma nativa como sus equivalentes configuraciones de un clúster virtualizado. Es decir, todos los escenarios fueron equivalentes en términos de uso de memoria total y número de unidades de procesamiento totales.

Esta prueba se realizó con el objetivo de analizar el comportamiento de dichos escenarios de clúster a lo largo del rango de memoria que son capaces de manejar, es decir, como se comportan los distintos escenarios con distintos tamaños de problema. Para ello se crearon distintas configuraciones de clúster viritual compuesto de 4 máquinas físicas como nodos de computación o contenedores de máquinas virtuales, según fuese el caso. De tal forma que una vez comprobados todos los escenarios posibles se pudiera establecer una comparativa entre los escenarios virtuales y el escenario nativo. Los escenarios a probar fueron:

- Un clúster físico compuesto por 4 máquinas físicas como nodos de computación con HT activado.
- Un clúster virtual compuesto por 4 nodos de computación virtuales, cada máquina virtual maneja 7960 MB de memoria y 8 cores lógicos. Cada máquina física aloja 1 máquina virtual.
- Un clúster virtual compuesto por 8 nodos de computación virtuales, cada máquina virtual maneja 3980 MB de memoria y 4 cores lógicos. Cada máquina física aloja 2 máquinas virtuales.
- Un clúster virtual compuesto por 16 nodos de computación virtuales, cada máquina virtual maneja 1990 MB de memoria y 2 cores lógicos. Cada máquina física aloja 4 máquinas virtuales.
- Un clúster virtual compuesto por 32 nodos de computación virtuales, cada máquina virtual maneja 995 MB de memoria y 1 core lógico. Cada máquina física aloja 8 máquinas virtuales.

Como se puede observar, todos los escenarios fueron equivalentes en términos de unidades de procesamiento[19] y memoria total[20]. Estas configuraciones fueron iguales que en la prueba V1 para el caso de tener *HyperThreading* activado y tal y como se mostró en la tabla 5. Se hace referencia a dichos escenarios como *Phys (con HT)*, *Virt 4 Nodos*, *Virt 8 Nodos*, *Virt 16 Nodos* y *Virt 32 Nodos*, respectivamente.

En esta prueba todos los clústers virtuales tuvieron HT activado, por lo que no se hace referencia a esta característica en cuanto a los clúster virtuales. El clúster físico de referencia también tuvo HT activado, ya que se persiguió el que fueran lo más parecidos posibles, y no hay equívoco posible en cuanto al número de nodos de computación del que se compone el clúster. No obstante, sí se hace mención a su condición de tener HT activado porque se comprobó también el comportamiento de dicho clúster físico de referencia con HT desactivado. Esto se hizo ya que se considera que puede tener un gran impacto en el rendimiento del clúster en términos generales desde un punto de vista más amplio dentro del ámbito de HPC. Es decir, para no perder la perspectiva de este aspecto. Por tanto, se añadió este escenario extra:

- Un clúster físico compuesto por 4 máquinas físicas como nodos de computaición con HT **desactivado** al que se le hará referencia como *Phys (Sin HT)*. No obstante, se recuerda que este escenario está añadido como extra y la prueba V2 está centrada en todos los escenarios con HT activado.

Para cada uno de los escenarios se probaron los mismos tamaños de problema N elementos de lado, tal y como se muestra en la tabla 6.

Para todos los escenarios, el tamaño de bloque NB se amplió respecto a las pruebas R1 y V1 al rango entre 32 y 1024 elementos, y para todos los escenarios con HT activado, es decir,

[19] 4 Máquinas x 8 Cores Lógicos/Máquina = 32 Cores Lógicos.
[20] 4 Máquinas x 7960 MB/Máquina = 31840 MB = 31.1 GB.

N	%Mem
10000	15.5
20000	31.0
40000	61.9
50000	77.4
50800	78.6

Tabla 6. Los tamaños de problema probados asociados al porcentaje del tamaño máximo de matriz que cabe en memoria principal del clúster.

los escenarios principales de la prueba, se probaron las rejillas de procesos PxQ 1x32, 2x16 y 4x8[21].

Es de esperar que el tamaño de bloque NB, en vista de lo obtenido en anteriores pruebas, influya en el rendimiento de forma que éste aumentea medida que aumente el número de elementos a enviar por mensaje (NB) y así minimizar el número de comunicaciones entre procesos. Esto es así hasta que se llegue a un punto en el que el rendimiento no solo crezca con el aumento de NB sino que decrezca, debido a que empieza a ser demasiado costoso manejar mensajes de tamaño tan grande.

Se esperaba obtener los mejores resultados con el escenario *Phys (Con HT)* en términos de rendimiento absoluto. Entre los clústers virtuales, en vista de lo obtenido en la prueba V1 y debido a que se minimizan las comunicaciones entre máquinas, se esperaba obtener los mejores resultados en el escenario *Virt 4 Nodos*. Para *Virt 8 Nodos* se preveía que su rendimiento fuera descendiendo a medida que creciese el tamaño del problema, siempre en relación al escenario *Phys (Con HT)*. Eso sería debido al sobrecargo de uso de memoria y uso de recursos de computación que representa la virtualización.

Era previsible obtener resultados con un rendimiento muy pobre en los escenarios *intermedios*, es decir, los escenarios *Virt 8 Nodos* y *Virt 16 Nodos*, en vista de los resultados obtenidos en la prueba V1[22].

Si se confirmase este comportamiento, será una prueba a favor de que en los escenarios *intermedios* se produjo un aumento de las comunicaciones varios niveles, que en combinación producen un mal rendimiento. Todas las máquinas físicas deben controlar varias máquinas virtuales, y a su vez todas las máquinas virtuales controlan varias unidades de procesamiento. Esto implica gran número de comunicaciones y procesos implicados en la coordinación de estas dos capas de abstracción que afectaría significativamente al rendimiento del clúster y que si el problema no es muy grande, no compensaría utilizar estos escenarios.

Por el contrario, si hubo mejoría en el rendimiento de los escenarios *intermedios Virt 8 Nodos* y *Virt 16 Nodos* al aumentar de tamaño de problema a resolver, entonces se confirmaría la existencia de un cuello de botella a nivel de las comunicaciones en dichos escenarios (gestión de las máquinas virtuales por parte del contenedor y las comunicaciones internas de las mismas) que puedieran resultar comparativamente excesivas en relación a problemas relativamente pequeños y verse compensados en problemas de mayor tamaño.

En el escenario *Virt 32 Nodos*, sin embargo, se produce el máximo de comunicaciones externas entre máquinas virtuales a nivel de los procesos relacionados con el problema. En este ecenario se puede eliminar el efecto de que una máquina virtual tenga que manejar varias unidades de procesamiento, pero a su vez, todas las comunicaciones entre procesos son comunicaciones externas entre máquinas virtuales. Por lo tanto, es de esperar que este escenario

[21] Para el escenario extra *Phys (Sin HT)* se probaron las rejillas de procesos más parecidas posibles: 1x16 como rejilla menos descompensada, 2x8, y 4x4 como rejilla más compensada.

[22] Pese a que dichos resultados no se reflejan hasta la siguiente sección 6. No obstante, sirva de adelanto que en la prueba V1 se obtienen pésimos resultados de rendimiento en los escenarios que implican múltiples máqinas virtuales por cada máquina física en los que cada máquina virtual cuenta con varias unidades de procesamiento.

proporcione una idea de hasta que punto son significativas las comunicaciones entre unidades de procesamiento de una misma máquina y lo que le cuesta al entorno virtual mantener el aislamiento de las máquinas virtuales en relación a los recursos HW que tiene que asignarles.

En cuanto a aspectos de la configuración de HPL a tener en cuenta, la rejilla de procesos PxQ que obtenga mejores resultados en cada escenario puede indicarnos si las comunicaciones entre procesos suponen un cuello de botella en la resolución del problema. Si un escenario dió mejores resultados con una rejilla más descompensada (1x32) significará que se resuelve el problema suponiendo que las procesos, y por extensión máquinas, se comunicaron mejor entre sí de forma *punto a punto* o *en línea*[23], minimizando de esta forma el número de canales de comunicación entre procesos relativos a la resolución del *benchmark*. Sin embargo, si resulta que se obtuvieron mejores resultados con una rejilla de procesos más compensada es de suponer que resulta más cómodo para resolver el problema un esquema en el que prime la fluidez de información entre procesos, esto es, con una rejilla 4x8 o incluso 2x16, aunque esta última pudiera parecer bastante descompensada.

5.4.3 Prueba en Clúster Virtual 3 (V3): Tamaño Estándar Por Distribución en Máquinas

El objeto de esta prueba reside en comprobar las diferencias entre un clúster virtual distribuido en varias máquinas físicas y un clúster virtual concentrado en una sola máquina física.

Como cada máquina física tiene 8 cores lógicos, se probaron los siguientes escenarios[24]:

- Un clúster físico con 1 máquina física como nodo de computación.
- Un clúster virtual con 1 máquina física como contenedor de máquinas virtuales. En ella se alojaron 8 máquinas virtuales.
- Un clúster virtual con 8 máquinas físicas como contenedores de máquinas virtuales. En cada una de ellas se alojó una única máquina virtual.

Cada una de las máquinas virtuales tiene 1GB de memoria RAM y un core lógico asignado como unidad de procesamiento.

En este escenario solo se probaron dos tipos de rejilla PxQ: 1x8 y 2x4, las cuales en este caso son la descompensada y la compensada, respectivamente. Se probaron tamaños de mensaje NB en el rango de los 32 a 1024 elementos y con dos tamaños de problema: *Problema de Tamaño Estándar* con N = 10000, y N = 20000.

En el escenario donde las 8 máquinas virtuales están distribuidas en 8 máquinas físicas distintas (escenario *Disperso*) se espera que en el apartado de las comunicaciones juegue en contra, ya que la red de interconexión entre máquinas físicas es más lenta que las comunicaciones internas en una misma máquina física.

El escenario donde las 8 máquinas virtuales están concentradas en una máquina física (escenario *Concentrado*) se verá perjudicado por el hecho de que las máquinas virtuales tienen que acceder a recursos únicos de la máquina física que las contiene a todas, como el acceso al disco duro, el bus de comunicaciones, o la tarjeta de red.

De forma que en este experimento se comprobó si es preferible tener ciertos elementos clave distribuidos a costa de emplear una red de interconexión o tener concentrados todos los sistemas a costa de que tengan que repartirse el acceso a los recursos. Dicho de otro modo, si es mejor el hecho de distribuir un clúster virtual en el máximo de máquinas físicas posibles o bien es preferible concentrarlo en el mínimo de máquinas físicas posibles.

[23] Cada proceso sólo se comunica con el/los proceso/s que manejen la porción de la matriz adyacente/s.
[24] Se da por supuesto que cada clúster cuenta con un *frontend* que no se describe.

5.4.4 Prueba en Clúster Virtual 4 (V4): Carga Constante Por Distribución en Máquinas

Esta prueba tuvo dos propósitos relativos al uso de la virtualización: evaluar el comportamiento de minimizar o maximizar el número de máquinas virtuales en un clúster, y comprobar como responde un clúster virtualizado al escalarlo horizontalmente. En cada caso se resolvió un problema que siempre representaba el mismo porcentaje de la memoria que manejaba cada clúster, y se probaron distintos tamaños de bloque NB entre 32 y 1024 elementos.

Para ello se probarón clústers formados por 1, 2 y 4 máquinas físicas con *HyperThreading* activado, tal y como se describen a continuación:

- Un clúster físico compuesto por 1, 2 y 4 máquinas físicas. Cada una de las máquinas físicas contó con 8 unidades de procesamiento y 8 GB de RAM.
- Un clúster virtual compuesto por 1, 2 y 4 máquinas físicas las cuales alojaron 1 máquina virtual por cada máquina fsica. Cada una de las máquinas virtuales contó con 8 unidades de procesamiento y 8 GB de RAM.
- Un clúster virtual compuesto por 1, 2 y 4 máquinas físicas las cuales alojaron 8 máquinas virtuales por cada máquina fsica. Cada una de las máquinas virtuales contó con 1 unidad de procesamiento y 1 GB de RAM.

Se escogieron tamaños de problema que representaran el 80% de la memoria principal ya que es una buena carga de trabajo para este tipo de problemas HPC. No obstante esto significa que cada clúster, según cuantas máquinas físicas lo componen, deberá afrontar resolver un problema con un N proporcional, tal y como se puede observar en la tabla 7:

# Máquinas	N
1	10240
2	14482
4	20480

Tabla 7. Los tamaños de problema probados en cada uno de los clúster según el número de máquinas físicas que lo componen.

Se probaron en cada caso dos tipos de rejilla de procesos PxQ, a las que llamaremos *compensada* y *descompensada* simplemente por comparación. En vista de los resultados obtenidos en pruebas anteriores [25], se evitó en la medida de lo posible el uso de una rejilla de procesos PxQ que suponga una comunicación *punto a punto*, es decir de tipo 1xA siendo A cualquier número entero. La tabla 8 muestra las rejilla de procesos PxQ, *compensada* y *descompensada*, para cada clúster según el número de máquinas físicas empleadas.

# Máquinas	PxQ compensada	PxQ descompensada
1	2x4	1x8
2	4x4	2x8
4	4x8	2x16

Tabla 8. Los tamaños de problema probados en cada uno de los clúster según el número de máquinas físicas que lo componen.

Se espera que el escenario que maximiza el número de máquinas virtuales se vea perjudicado porque todas las comunicaciones realizadas entre procesos implican comunicaciones entre distintas máquinas virtuales.

[25] Tal y como se verá en la sección 6.

6 Resultados

En esta sección se procederá a describir los resultados obtenidos en las pruebas comentadas en la sección anterior. Se seguirá el mismo orden en el que fueron expuestas en la sección 5: primero se describirán los resultados obtenidos en las pruebas Cx acerca de configuración del clúster; posteriormente se analizarán los resultados obtenidos en la prueba R1 que trató los límites físicos del clúster; y por último se examinarán los resultados de las pruebas Vx, las centrales de este estudio, en las cuales se comprueba el efecto de la virtualización sobre el clúster.

6.1 Resultados Pruebas Configuración

Recordemos que cada una de las pruebas de configuración correspondía a cada uno de los atributos a definir en el clúster: la inerconexión de red (C1), el método de alojamiento de los procesos por parte del gestor de tareas SGE (C2), la elección de la bibliotecas de álgebra lineal a utilizar por HPL (C3), y el algoritmo que utilizará HPL para la difusión de los paneles (C4).

- **Inerconexión de Red (Prueba C1)**:

  Como era de esperar, la configuración de clúster con Ethernet Gigabit resultó significativamente mejor que con Ethernet 10/100.

  El resultado obtenido con la red Ethernet 10/100 estuvo muy restringido en el rango de los 30-35 GFLOPS. Aún realizando otras variaciones en el entorno como por ejemplo variar el tamaño del problema, aumentando/disminuyendo el número de nodos de computación, o activando/desactivando *HyperThreading* en dichos nodos, y con otras confifuraciones probadas en este conjunto de pruebas Cx e incluso bajo virtualización no se pudo superar un rendimiento de 35 GFLOPS.

  Utilizando Ethernet Gigabit se obtuvieron limites de rendimiento entre 30 y 80 GFLOPS, lo cual supera fácilmente el rendimiento obtenido usando Ethernet 10/100, pero con gran variación de los mismos dependiendo otros factores.

  No obstante, y como ya se dijo, puede resultar interesante ver la experiencia con la red más lenta para prever su comportamiento en entornos en los que solo se disponga de dicha interconexión.

  Esto tiene otra lectura la cual es que en caso de solo disponer de una interconexión Ethernet 10/100 se puede hacer uso de un clúster virtualizado sin temor a pérdidas de rendimiento debido a la virtualización, ya que el cuello de botella en el rendimiento que supone tal tipo de interconexión hace que se enmascare las pérdidas de rendimiento por virtualización y no supone desventaja alguna. Esto supone un resultado interesante desde un punto de vista práctico, pero sin mayor trascendencia en el ámbito teórico.
- **Alojamiento de Procesos (SGE) (Prueba C2)**:

  Al no haber información de que tipo de alojamiento de procesos puede resultar más conveniente entre las opciones que proporciona el gestor de tareas SGE, se prueban y se comparan ambos métodos *Fill Up* y *Round Robin*, arrojando resultados muy similares entre ambos métodos con una ligera ventaja de rendimientos a favor de *Fill Up*, aunque no significativa.

  Por lo que se decide utilizar para el resto de pruebas *Fill Up*[26] en detrimento de *Round Robin*, método que tiende a repartir los procesos entre las distintas máquinas de la forma más equitativa posible.

[26] Recordemos que con *Fill Up* los procesos se reparten de forma que se asignan todos los procesos a una misma máquina, y en caso de que haya más procesos se asignan todos los procesos restantes a otra máquina, y así sucesivamente hasta que no quedan más procesos por asignar.

– **Bibliotecas de Álgebra Lineal (Prueba C3)**:

Al comparar la biblioteca OpenBLAS con GotoBLAS2 de cara a ver cual puede ser más adecuada para ejecutar HPL, los resultados obtenidos han sido similares en ambos casos sin poder hallar diferencias significativas entre ambas, aunque con una ligerísima ventaja a favor de GotoBLAS2.

Esto no significa que la biblioteca OpenBLAS sea peor, si no que quizás para el benchmark HPL no supone diferencia[27], o que no es significativo para este tipo de problemas debido al tamaño del mismo, por número de nodos, capacidad de computacion, o alguna otra razón.

Más allá de comprobar el correcto funcionamiento de ambas bibliotecas, no se puede realizar ninguna conclusión al respecto. En pruebas posteriores utilizará la biblioteca GotoBLAS2 al comprobarse una ligera mejoría con dicha biblioteca aunque no sea una diferencia signigicativa, y contar con más respaldo documental en lo que respecta a este ámbito de estudio. No obstante, se invita a utilizar la biblioteca OpenBLAS en futuros estudios.

– **Difusión del Panel (HPL) (Prueba C4)**:

Se probaron los métodos de difusión de paneles HPL te tipo *Ring* y *Long* debido a que la documentación original recomienda una difusión de panel tipo *Long* en caso de que la red de interconexión interna sea significativamente más lenta que la capacidad de cómputo, proporicionalmente hablando[28].

Pese a eso se obtuvieron resultados 3 veces más lento con el método *Long* que con el método *Ring*. Es por esto por lo que se decide utilizar *Ring* como método de difusión de los paneles en HPL en todas las pruebas restantes y, por lo tanto, no cambiar la configuración por defecto.

6.2 Resultado Prueba R1

En la prueba R1 se comprobó cual era el tamaño máximo de problema que el clúster era capaz de resolver, sin utilizar virtualización pero sí se probó si el uso o no de *HyperThreading* era apropiado para estos escenarios.

Los resultados obtenidos en la prueba fueron similares a los mostrados en la figura 4, la cual muestra los resultados de las pruebas de las rejillas de procesos PxQ más compensada.

Cada una de las curvas representa un tipo de clúster y un tamaño de problema a resolver. Las curvas anaranjadas representan los resultados del clúster sin utilizar *HyperThreading*, y las azuladas cuando se utilizó *HyperThreading*. Cada una de las curvas de un color similar (azulado o anaranjado) representa un tamaño de problema N distinto, variando desde matrices a resolver con 10000 elementos por lado hasta los 60000 elementos por lado.

Los rendimientos obtenidos, medidos en GFLOPS, se representan en el eje de las ordenadas. Mientras que el eje de ordenadas sirve para indicar con que tamaño de bloque NB se probó el clúster y tamaño de prolema N correspondiente. Se recorren los tamaños de bloque *NB* de 32 a 256 elementos por mensaje.

Aunque la gráfica mostrada en la figura 4 solo muestra una de las tres variaciones de rejilla utilizadas para cada configuración, la rejilla PxQ más compensada, se han obtenido resultados similares en las pruebas con rejilla de procesos PxQ más descompensadas. Todos los resultados obtenidos en esta prueba se pueden observar en el anexo correspondiente a los resultados de la prueba R1 en la página 53 (Anexo E, Datos relativos a la prueba R1). Por lo que aunque lo explicado a continuación se centre en el caso de una rejilla más compensada, no solo se tratará de explicar ese caso.

[27] HPL es de 2012, la última actualización de GotoBLAS2 es de 2010, y OpenBLAS evoluciona del anterior pero conserva código del mismo.

[28] Tal y como creemos que es el caso al utilizar una red Ethernet Gigabit en vez de una tipo Infiniband por ejemplo, que suele ser el estándar en estos casos.

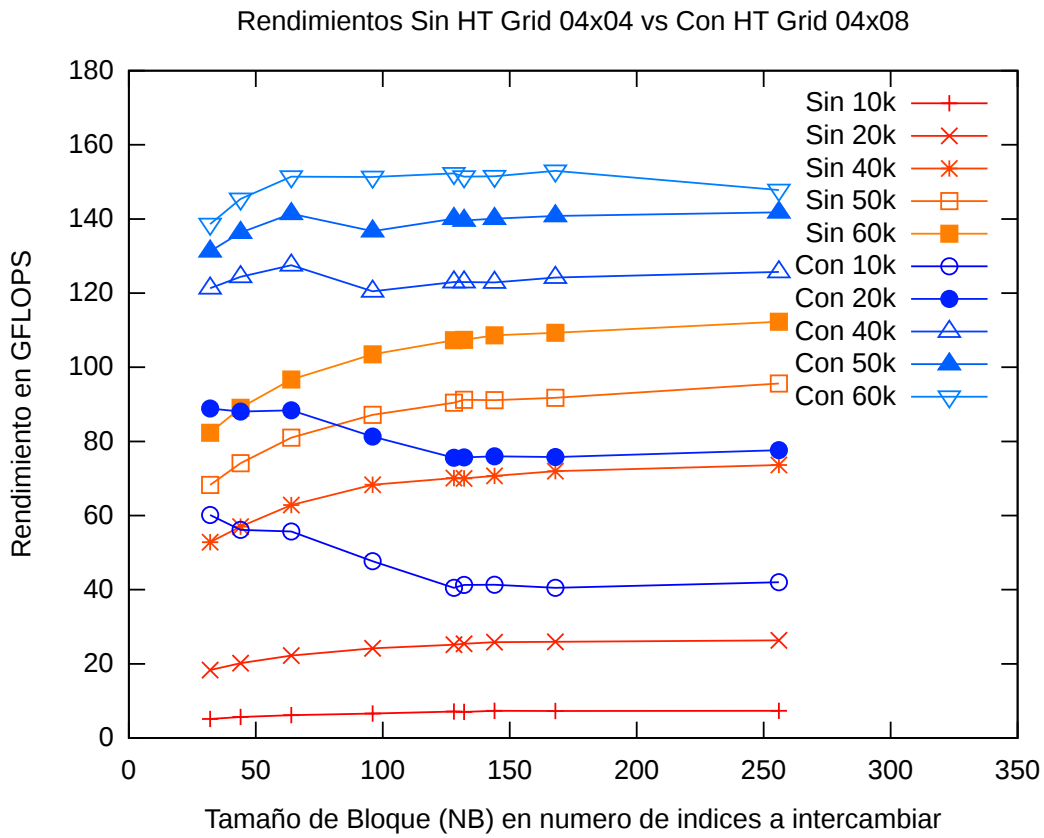

Fig. 4. Rendimiento obtenido para dos casos (con *HT* y sin HT) al resolver problemas variando el tamaño (entre 10K y 60K). En la configuración con *HT* desactivado la rejilla de procesos es de 4x4, y en la configuración con *HT* desactivado la rejilla de procesos es de 4x8. El eje X representa la variación de tamaño de bloque NB (entre 32 y 256 elementos) y a lo largo del eje Y se muestran los rendimientos en GFLOPS para cada una de las pruebas.

Como primer punto a destacar, por ser la segunda motivación de esta prueba y por resultar más evidente, es el uso de *HyperThreading*. Se resuelven las dudas acerca de la conveniencia de su uso ya que se obtuvieron mejores resultados haciendo uso de *HyperThreading*, del orden casi del doble de rendimiento que lo obtenido sin hacer uso del dicha tecnología. De forma que se zanja la cuestión y se establece el uso de *HyperThreading* en el resto de pruebas, que por otra parte son las que implican el uso de virtualización.

De hecho, con *HyperThreading* los resultados son más estables y más parecidos a lo predicho en la tabla 4 de la página 17. Por otra parte hay que destacar que la eficiencia (rendimiento por unidad de procesamiento) es menor utilizando *HyperThreading*, pero este estudio busca el máximo rendimiento bruto con los medios disponibles y no la máxima eficiencia.

Si nos fijamos en el rendimiento de los clústers según la variación en el tamaño del bloque NB parece que para el caso de tener *HyperThreading* activado hay un óptimo de NB en torno a los 128-132 elementos. En caso de tener el *HyperThreading* desactivado el tamaño de bloque no parece haber llegado a su tamaño óptimo en el rango probadom, aunque parece acercarse a su límite ya que en torno a los 256 elementos de NB el rendimiento prácticamente ha dejado de crecer, motivo por el cual se aumenta el rango de NB en las siguientes pruebas.

Como ya se dijo, en esta sección solo se muestra el resultado de los rendimientos de la rejilla de procesos PxQ más compensada: [29]. No obstante, para el resto de rejillas probadas se obtuvieron resultados similares exceptuando que no el sistema no fue capaz de resolver los problemas de mayor tamaño con estas rejillas más descompensadas: ni los que se acercan al máximo teórico, ni los de tamaño 60000 para tamaños de bloque grande. Véase el anexo E, *Datos relativos a la prueba R1* en la página 53 para ver todos los resultados de la prueba R1.

[29] 4x8 en el caso de que *HyperThreading* esté activado y 4x4 en caso de que no lo esté.

Aunque estos escenarios con esta rejilla de proceso no hayan alcanzado el límite previsto no significa que sea un error o un fracaso, simplemente se tiene un conocimiento aproximado acerca de donde esta el limite de tamaño del problema para el sistema empleado, lo cual era uno de los objetivos del experimento.

Los tamaños de problema entorno a los 50000 elementos parcen explotar el máximo rendimiento, lo cual confirma la sugerencia de que N al 80% del límite de la capacidad del clúester es ideal para este tipo de problemas. Exceptuando los casos antes mencionados, y a rasgos generales, la prueba ha resultado tal y como se esperaba ya que los rendimientos obtenidos han coincidido con lo previsto en la tabla 4 en la página 17, especialmente cuando *HyperThreading* ha estado activado.

Continuando con el aspecto de la rejilla de procesos PxQ podemos confirmar, tal y como se sugiere en la documentación original de HPL, que por regla general se prefieren las rejillas más compensadas ya que los casos de rejilla 2x16 y 1x32[30] son peores en su conjunto que los de la rejilla más compensada.

No obstante hay excepciones. Por ejemplo, para N igual a 50000 con *HyperThreading* en el caso de las rejillas 2x16 y 4x8 donde es mejor el caso de la rejilla algo descompensada. También en el caso de tener desactivado *HyperThreading* que la rejilla más descompensada fue la que obtuvo mejores resultados.

Por finalizar con las excepciones, sin *HyperThreading* y usando la rejilla PxQ 2x8 no produce resultados ni siguiera para tamaño 50000. Es curioso ya que N igual a 50000 es un límite de tamaño relativamente bajo como para que no sea capaz de resolverlo.

6.3 Resultado Prueba V1

La prueba V1 recordemos que consistía en ejecutar el *Problema de Tamaño Estándar* en un clúster físico y en todos los clústers virtuales posibles utilizando el mismo número de unidades de procesamiento y memoria principal. Los escenarios virtuales probados fueron cuatro: el que minimiza el número de máquinas virtuales (o *réplica virtual del clúster*, donde cada máquina física asigna todos sus recursos a una única máquina virtual), el escenario que maximiza el número de máquinas virtuales (cada unidad de procesamiento de las máquinas físicas del clúster es asignada a una única máquina virtual), y dos escenarios *intermedios* (donde cada máquina física aloja varias máquinas virtuales las cuales tienen asignadas varias unidades de procesamiento).

Los resultados obtenidos se muestran en las figuras 5 y 6. En ellas se muestra el mejor resultado, medido en GFLOPS, obtenido por cada escenario. En ambas figuras se muestran los mismos resultados vistos de diferente modo: en la figura 5 se muestran los resultados de cada escenario clúster probado agrupados por las rejillas PxQ utilizadas, y en la figura 6 se pueden observar los resultados según las rejillas PxQ probadas agrupadas por escenario clúster utilizado. Todos los resultados obtenidos en esta prueba se pueden observar en el anexo correspondiente a los resultados de la prueba V1 en la página 54 (Anexo F, Datos relativos a la prueba V1).

Si nos fijamos en la figura 5, podemos apreciar mejor los resultados según el tipo de clúster empleado. Es interesante observar que el escenario nativo obtuvo peores resultados de rendimiento respecto a su *réplica* virtual [31].

En cuanto al resto de escenarios podemos concluir que los escenarios *intermedios* obtuvieron un pésimo resultado en cuanto al rendimiento obtenido en comparación con el escenario nativo. El escenario que maximiza el número de máquinas virtuales, pese a no tener unos buenos resultados, en torno al 50% y dependiendo de las circustancias de entre un 25% y un 75% del

[30] 2x8 y 1 x16 para los casos en los que *HyperThreading* esté desactivado.

[31] Se entiende por *réplica* el escenario virtual en el que hay el mismo número de máquinas físicas que virtuales ya que a cada máquina virtual se le asignan todos los recursos posibles de una única máquina física.

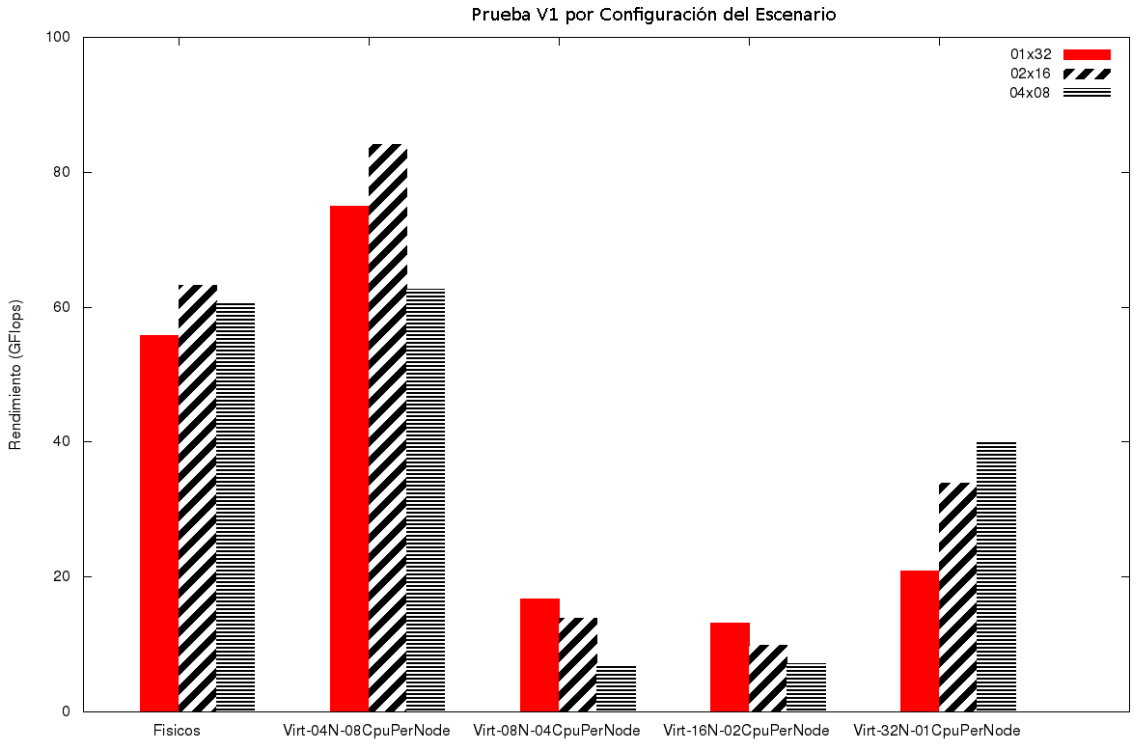

Fig. 5. Rendimientos obtenidos en la prueba V1 para cada una de las rejillas de procesos probados, agrupados en el eje X por cada uno de los escenarios utilizados (configuración nativa del clúster y las distintas configuraciones virtuales). El eje Y indica el rendimiento obtenido en GFLOPS.

rendimiento del escenario nativo, puede ser útil en ciertos casos[32], ya que aunque los resultados comparados con el entorno nativo y con la *réplica* virtual son peores, la eficiencia en este escenario es mayor y son configuraciones usables.

Si nos fijamos en la figura 6 podemos observar mejor la influencia de la rejilla PxQ en el rendimiento de los clústers. Este es un parámetro importante ya que es el único que diferencia las ejecuciones de HPL entre escenarios [33], obviando la naturaleza nativa o virtual del clúster y la distribucuión de recursos en los clústers virtuales. Además sabemos que las comunicaciones se ven afectadas por este parámetro por lo que puede ser bastante influyente, y de hecho lo es.

El escenario con más máquinas virtuales se cumple la teoría en cuanto a medida que se compensa la rejilla PxQ se obtiene mejor rendimiento pero esto no ocurre con todos los escenarios. De hecho en los escenarios virtuales *intermedios*, en contraposición con la teoría, el rendimiento aumenta a medida que la rejilla PxQ está más descompensada. Tampoco se cumple en los casos del escenario nativo y la *réplica* virtual, los cuales parecen tener un comportamiento similar aunque de forma más pronunciada en el caso del escenario virtual y una diferencia: mientras que en el escenario nativo si parece seguir la tendencia que marca la teoría de que una rejilla más compensada da mejores resultados, ya que la rejilla más compensada solo es ligeramente peor que la rejilla intermedia, con la *réplica* virtual, utilizando la rejilla más compensada, se obtuvieron con diferencia los peores resultados en ese escenario.

Estas diferencias de rendimiento debido a la utilización de distintas rejillas de proceso PxQ no parecen ser muy significativas en el caso de utilizar el clúster en su forma nativa pero sí en el caso de los clústers virtuales. Estas pérdidas de rendimiento dentro de un mismo escenario virtual se pueden cuantificar en torno al 20-25% en el caso de utilizar un mínimo de máquinas virtuales y casi del 50% en el caso del escenario donde se maximizan.

[32] Como en *IaaS* o *PaaS*.
[33] N, NB y el método de difusión de panel son iguales para todos los escenarios a nivel de HPL

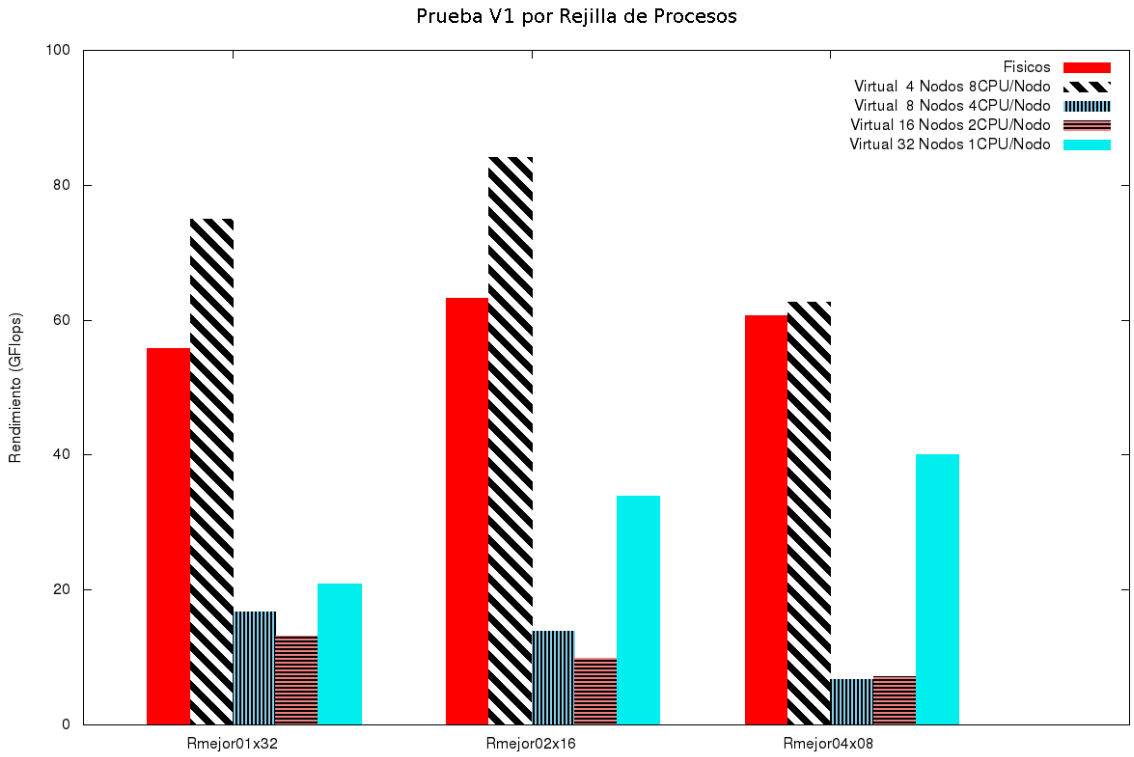

Fig. 6. Rendimientos obtenidos en la prueba V1 para cada uno de los escenarios probados (configuración nativa del clúster y las distintas configuraciones virtuales), agrupados en el eje X por cada una de las rejillas de procesos utilizadas. El eje Y indica el rendimiento obtenido en GFLOPS.

Recordando que el problema consisitió en la resolución de un problema relativamente pequeño para la capacidad del clúster, se pueden obtener las siguientes conclusiones provisionales:

- Las rejillas más compensadas no siempre dan los mejores resultados, y un clúster de naturaleza virtual parece ser mucho más sensible en términos de rendimiento a la variación de la rejilla de procesos PxQ.
- Se puede observar que hay casos en los que se pueden conseguir mejores resultados con un clúster virtual que con uno nativo o real. Concretamente parece que se consigue con aquellas configuraciones virtuales que representan directamente la infraestructura física, es decir, una *réplica* del clúster físico utilizando contenedores virtuales. Esta puntualización acerca del tipo de virtualización es importante ya que *ataca* directamente al procesador, por lo que es especialmente propicia para este propósito.
- Contrariamente a la idea inicial, no es el escenario virtual con más número de máquinas la que ha ido más lenta, sino los escenarios *intermedios*, es decir, en los que cada máquina física aloja varias máquinas virtuales y cada una de ellas con varias unidades de procesamiento.
- Por lo tanto, en cuanto a configuraciones virtuales, parece ser que solo merecen la pena dos de ellas, las más extremas. Por su rendimiento bruto, la *réplica* virtual que imita lo máximo posible la configuración de la infraestructura física y por lo tanto minimiza el número de máquinas virtuales, y la configuración que maximiza el número de máquinas virtuales posibles por el número de nodos disponibles en el clúster. Por lo tanto, el potencial uso de una u otra configuración dependerá de en que ámbito se desea aplicar.
- El escenario que maximiza el número de máquinas virtuales posibles obtiene una pérdida en el rendimiento que pese a ser significativo puede llegar a ser asumible. Se obtienen pérdidas de rendimiento entre un 25-75% por lo que si interesa multiplicar el número de equipos disponibles puede ser una opción aceptable.

– El mal comportamiento de los escenarios *intermedios* podrían explicarse debido a que las comunicaciones se maximizan. Esto es debido a que hay una cantidad considerable de comunicaciones entre distintas máquinas virtuales alojadas en una misma máquina virtual. Estas comunicaciones serán tratadas por las máquinas virtuales como comunicaciones al exterior y sin embargo no pasan por la red de interconexión interna del clúster haciendo que la máquina física tenga que resolverlas aumentando la carga computacional y por lo tanto reduciendo el rendimiento.

6.4 Resultado Prueba V2

Esta prueba V2 fue similar a la V1 en su planteamiento aunque aumentando el tamaño de los problemas a resolver hasta llegar al límite del sistema.

Tiempo y rendimiento obtenido por el mejor resultado de cada configuración y tamaño de problema probado en los escenarios nativos

Tamaño de Lado N	Phys (Sin HT) 4 Nodos	Phys (Con HT) 4 Nodos
10000 coef.	13.8s 48.4 GFlops	10.2s 65.4 GFlops
20000 coef.	65.9s 81.0 GFlops	51.9s 102.8 GFlops
40000 coef.	363.9s 117.3 GFlops	295.3s 144.5 GFlops
50000 coef.	644.0s 129.4 GFlops	538.2s 154.9 GFlops
50800 coef.	691.7s 126.4 GFlops	560.1s 156.0 GFlops

Tabla 9. En cada celda, a la izquierda se muestra el tiempo en segundos obtenido por el mejor resultado (para esa configuración y tamaño de problema) y a la derecha el rendimiento en GFlops para ese mismo resultado.

Tiempo y rendimiento obtenido por el mejor resultado de cada configuración y tamaño de problema probado en los escenarios virtuales

Tamaño de Lado N	Virt 4 Nodos	Virt 8 Nodos	Virt 16 Nodos	Virt 32 Nodos
10000 coef.	10.4s 64.3 GFlops	39.1s 17.1 GFlops	30.9s 21.6 GFlops	20.3s 32.9 GFlops
20000 coef.	47.5s 112.3 GFlops	141.5s 37.7 GFlops	144.3s 37.0 GFlops	103.6s 51.5 GFlops
40000 coef.	422.9s 100.9 GFlops	486.3s 87.8 GFlops	615.3s 69.4 GFlops	506.6s 84.2 GFlops
50000 coef.	1026.5s 81.2 GFlops	1013.6s 82.2 GFlops	1006.1s 82.8 GFlops	1122.0s 74.3 GFlops
50800 coef.	1017.4s 85.9 GFlops	1011.4s 86.4 GFlops	1062.7s 82.2 GFlops	- -

Tabla 10. En cada celda, a la izquierda se muestra el tiempo en segundos obtenido por el mejor resultado (para esa configuración y tamaño de problema) y a la derecha el rendimiento en GFlops para ese mismo resultado.

Rejilla de Procesos y NB del mejor resultado de cada configuración y tamaño de problema probado

Tamaño de Lado N	Phys (Sin HT) 4 Nodos	Phys (Con HT) 4 Nodos	Virt 4 Nodos	Virt 8 Nodos	Virt 16 Nodos	Virt 32 Nodos
10000 elementos	01x16 448	02x16 32	01x32 96	01x32 168	02x16 512	04x08 44
20000 elementos	01x16 512	02x16 384	01x32 128	02x16 858	02x16 1024	04x08 730
40000 elementos	01x16 512	02x16 384	02x16 512	02x16 858	02x16 730	04x08 858
50000 elementos	01x16 448	02x16 320	04x08 64	01x32 144	02x16 384	04x08 64
50800 elementos	01x16 512	02x16 320	04x08 96	04x08 512	04x08 320	- -

Tabla 11. En cada celda, a la izquierda se muestra la rejilla de procesos empleada por el mejor resultado (para esa configuración y tamaño de problema) y a la derecha el tamaño de bloque NB, expresado en elementos de la matriz.

En las tablas 9 y 10 se muestran, para cada escenario y tamaño de problema, el mejor tiempo obtenido y el rendimiento en GFLOPS que se obtuvieron. Los mejores resultados obtenidos para cada escenario y tamaño de problema se recogen en la tabla 11 donde se muestran que rejilla PxQ y que tamaño de bloque NB fue usado para obtener el mejor resultado.

Al pasar a tamaños de problema mayores y haciendo una primera lectura de los mejores resultados obtenidos en cada escenario recogidos en la tablas 11, 9 y 10 se pueden hacer destacar respecto a los resultados esperados las siguientes observaciones:

- Efectivamente, y por regla general, el mejor resultado obtenido para cada tamaño de problema se corresponde al experimentado en el escenario *Phys (Con HT)*, es decir, el entorno nativo. Por el tipo de rejilla PxQ en este escenario nativo parece que el rendimiento permanece estable en un tipo de rejilla algo descompensada en consonancia con lo obtenido en la prueba V1.
- Se confirma que entre los clústers virtuales, el escenario que minimiza el número de máquinas virtuales es el que mejor resultados obtiene. De hecho, esta *réplica* virtual del escenario nativo ha resultado ser mejor en términos de rendimiento que la propia configuración nativa del clúster en algunas pruebas.
- Se repite el comportamiento mostrado en la prueba V1 para los escenarios *intermedios* pero se observa mejoría relativa respecto al rendimiento obtenido por el resto de escenarios de clústers virtuales a medida que aumenta el tamaño del problema. Dicho comportamiento es en parte explicable debido a la relación entre el tamaño del problema y la infraestructura software que tiene que manejar el sistema, ya que puede no compensar el sobrecoste de dicha capa de virtualización y de las comunicaciones entre unidades de procesamiento y máquinas virtuales, en comparación al tamaño problema. No obstante, estas diferencias de rendimiento utilizando escenarios *intermiendios* se reducen a medida que aumenta el tamaño de problema, siendo casi preferible utilizar un escenario en el que una máquina virtual disponga de varias unidades de procesamiento, sin llegar al extremo de tener todas, cuando el tamaño del problema sea cercano al límite de la capacidad del clúster. Esto se puede observar en las tablas 9 y 10 para los valores de N igual a 50000 y 50800.
- El escenario *Virt 32 Nodos* ha resultado ser el más regular en su comportamiento, manteniendo unas diferencias de rendimiento estables en términos relativos al rendimiento obtenido por el clúster físico en ese mismo problema. Estas diferencias se pueden cuantificar en torno al 50% del rendimiento obtenido por el entorno nativo. No obstante, esta configuración virtual en la que se maximiza el número de nodos es incapaz de resolver problemas relativamente grandes.

En la figura 7 se puede ver el comportamiento de los mejores resultados obtenidos por cada escenario en relación al escenario nativo *Phys (Con HT)*. En dicha figura se pueden apreciar mejor varias cosas, como que los rendimientos relativos tienden a concentrarse en torno al mantenido por el escenario con el máximo de máquinas virtuales, aunque este no llegue al límite del tamaño de problema propuesto.

También se puede apreciar en la figura 7 para que rangos de uso de memoria puede resultar apropiado el uso de un clúster virtual para la resolución del problema, y que tipo de clúster virtual compensa en cada caso.

Por ejemplo, se ve que para tamaños relativamente pequeños de problema directamente compensa hacer uso de la *réplica* virtual del clúster físico. Sin embargo, a medida que aumenta el tamaño de problema y en vista de las pocas diferencias existentes en el tiempo empleado para resolver el mismo puede compensar el uso de otro tipo de clúster virtual, si se desea maximizar el número de nodos, o bien puede no interesar en absoluto hacer uso de virtualización, si se desea maximizar el rendimiento bruto del clúster.

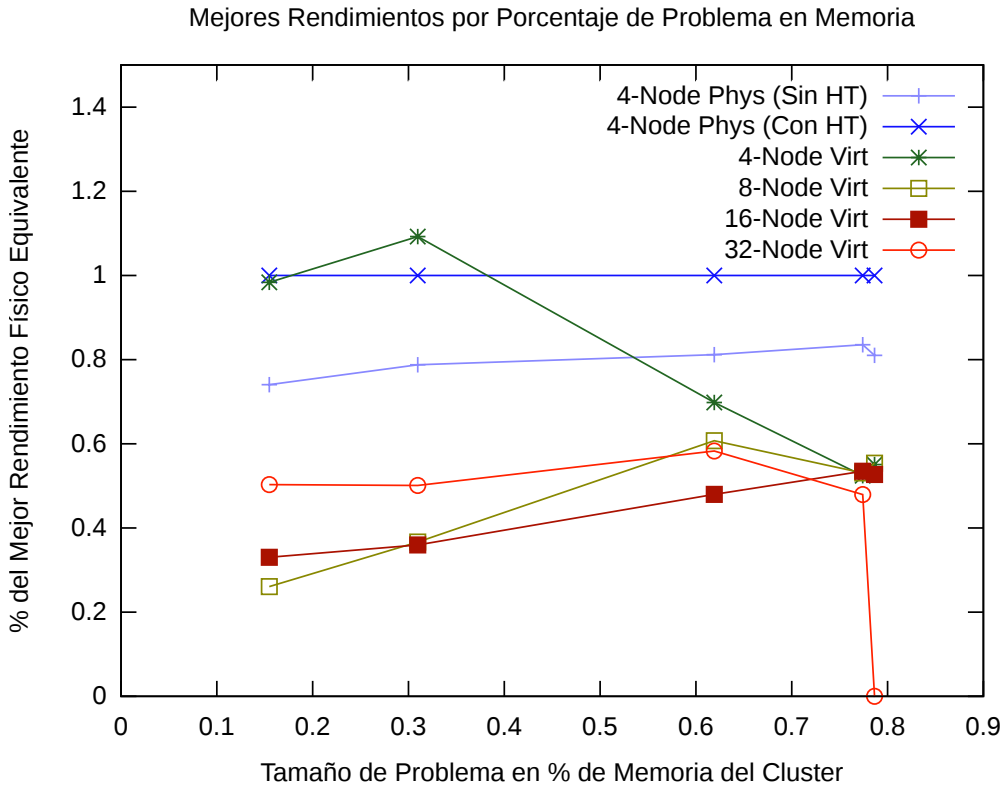

Fig. 7. Mejores rendimientos obtenidos por cada configuración del clúster probada (por cada uno de los tamaños de problema utilizados, expresados en porcentaje de memoria principal del clúster utilizada) en base al porcentaje del rendimiento obtenido por el clúster físico para el mismo tamaño de problema.

Todos los resultados obtenidos en esta prueba se pueden observar en el anexo correspondiente a los resultados de la prueba V2 en la página 55 (Anexo G, Datos relativos a la prueba V2).

El hecho de que se confirmen los resultados esperados en cuanto al rendimiento ya que:

- La *réplica* virtual es el escenario el más próximo en rendimientos a los obtenidos por el escenario nativo.
- Los escenarios *intermedios* mejoran su rendimiento relativo en relación al resto de escenarios conforme aumenta el tamaño del problema. Lo que avala la hipótesis de que el número de comunicaciones en estos escenarios no compensa en problemas de tamaño pequeño-mediano.
- El escenario que maximiza el número de máquinas virtuales supone un indicador de rendimiento de cualquier escenario al llegar a los límites de uso de memoria principal.

Unido a las siguientes observaciones acerca del comportamiento de la rejilla de procesos PxQ:

- El escenario *Virt 4 Nodos* se puede observar un comportamiento inusual en la rejilla de procesos más descompensada (1x32) en los casos que resulta ser mejor o solo ligeramente peor que el entorno nativo, ya que en problemas de tamaño pequeño/mediano se minimizan las comunicaciones entre procesos.
- Sin embargo el uso de esta rejilla 1x32 resulta ser el más inestable ya que tiende a producir peores resultados o incluso no poder resolver el problema. Esto es especialmente cierto para tamaños grandes de problema y/o de bloque (NB), incluyendo el escenario *Virt 4 Nodos*.
- A medida que aumenta N se tiende a obtener mejores resultados con rejillas PxQ más compensadas. Aunque en términos generales los rendimientos obtenidos en las rejillas 2x16 y 4x8 son parejos y suele ser ligeramente mejor la rejilla algo descompensada, es decir, 2x16.

– El impacto y comportamiento de NB a lo largo de cada configuración ha sido lo esperado, con cierta tendencia hacia el uso de mayores tamaños de bloque para comunicarse en los clústers virtuales, una vez más hace pensar que se tiende a minimizar el número de comunicaciones hasta cierto límite.

Sin olvidar que debido en los factores que influyen en a la sincronía necesaria entre los procesos para que el *benchmark* HPL se ejecute lo mejor posible, hacen que se llegue a las siguientes conclusiones provisionales:

– A mayor número de máquinas virtuales, mayor número de comunicaciones entre nodos del clúster. Estas comunicaciones son gestionadas por el contenedor de máquinas virtuales y por el *frontend*, por lo que a mayor número de máquinas virtuales menos fluidez en la sincronía de procesos y peor rendimiento se obtendrá por regla general.
– A mayor tamaño de problema, proporcionalmente mayor numero de comunicaciones entre procesos[34]. Esto es común a todos los escenarios y es inherente al tipo de problema HPC a resolver, por lo que es necesario ser consciente de este comportamiento y discernirlo del impacto de la virtualización.
– En cualquier caso, tanto en un entorno virtual como en uno real, se tenderá a minimizar el número de comunicaciones entre procesos que permitan mantener una sincronía adecuada para la resolución del problema. Por ello, el rendimiento del sistema se verá perjudicado en los escenarios en los que se maximice las comunicaciones entre unidades de procesamiento de un mismo nodo de comunicacion, las comunicaciones entre máquinas virtuales y las comunicaciones entre distintos nodos de computación. Este es el caso de los escenarios *intermedios* donde en una misma máquina física coexisten varias máquinas virtuales con multiples unidades de procesamiento cada una.
– El escenario que implica el máximo de máquinas virtuales supone, en igualdad de condiciones de problema, el máximo de comunicaciones entre procesos. Todo escenario alternativo con el que se obtengan peores resultados será achacable a sobrecarga en la gestión de máquinas virtuales, y por lo tanto asumir que es el impacto de la virtualización en la falta de sincronía.
– A medida que se llegan a valores límite de uso de memoria se tiende a saturar el sistema, de forma que las diferencias de rendimiento entre distintos escenarios tienden a equipararse. En tal caso si podría ser interesante el uso de un escenario *intermedio* o que maximice el número de máquinas virtuales por la posibilidad de contar con más nodos, aunque a costa de una pérdida de rendimiento importante.

Esto explicará a partir de las hipótesis realizadas y de los resultados obtendidos, porqué cuando es necesario un menor número de comunicaciones, como es el caso del escenario *Virt 4 Nodos*, con una comunicación *punto a punto* se obtienen tan buenos resultados:

En este escenario cada contenedor de máquinas virtuales da el control de todos los recursos a una única máquina virtual por lo que no tiene que gestionar distintas máquinas con la sobrecarga que conlleva el aislamiento y sincronía entre las mismas y los recursos del sistema. Así que se dan mejores resultados con problemas que no impliquen uso intensivo de las comunicaciones y de los recursos de computación para gestión de las mismas, cosa que se da en problemas de hasta medio tamaño. A partir de aquí empiezan todos los escenarios a equilibrarse siendo el primero en fallar el escenario que implica el máximo de máquinas virtuales, debido al alto grado de exigencia de los equipos.

[34] En una relación al cuadrado del lado del problema N.

6.5 Resultado Prueba V3

La prueba V3 trata de comparar el comportamiento de un clúster virtual, compuesto por el máximo de máquinas virtuales posibles, distribuido en varias máquinas físicas en contraposición a concentrar las máquinas virtuales en una única máquina física.

En la figura 8 se muestran agrupados por escenario las medias de rendimientos según el NB utilizado para los dos tamaños de problema y las dos rejillas PxQ utilizadas.

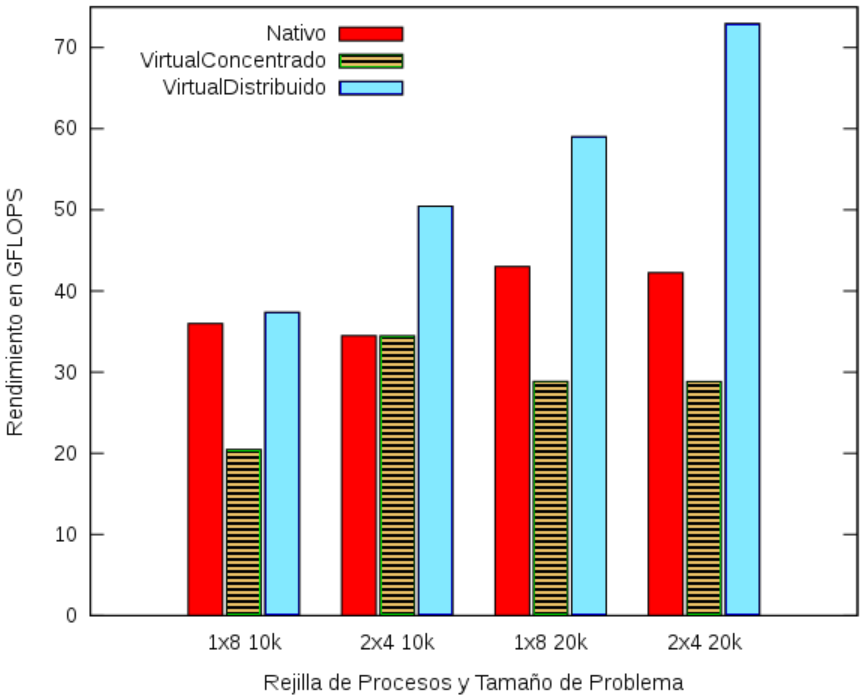

Fig. 8. Medias de rendimientos comparados de clústers comparando si dicho clúster se encuentra configurado por medio de realmente una única máquina física (Nativo), una unica máquina física como contenedor del máximo de máquinas virtuales posibles (VirtualConcentrado), o bien distribuyendo las distintas máquinas virtuales en distintas máquinas físicas (VirtualDistribuido). Todos los resultados están agrupados en el eje horizontal según la combinación de tamaño de problema resuelto y de rejilla de procesos PxQ empleada.

Todos los resultados obtenidos en esta prueba se pueden observar en el anexo correspondiente a los resultados de la prueba V3 en la página 60 (Anexo H, Datos relativos a la prueba V3).

Si se presta atención a los casos del escenario nativo en la figura 8 se puede ver que, en términos generales, no parece muy significativo el impacto de las comunicaciones debido a la rejilla de procesos. No hay grandes diferencias de rendimiento entre las dos PxQ probadas dentro de un mismo tamaño de problema, con una ligera ventaja cuando se utiliza una comunicación *punto a punto* entre procesos.

Las dos primeras agrupaciones de resultados en en la figura 8 se corresponden con un *Problema de Tamaño Estándar* y, en lo que respecta a los clústers virtuales, sí que parece que la distribución de procesos juegue un papel importante. En ambos casos se gana mucho rendimiento utilizando una comunicación entre procesos no lineal.

A medida que aumenta el tamaño de problema a resolver, estas comunicaciones se incrementan y se hacen más grandes las diferencias entre distintos escenarios del clúster pero no entre rejillas de procesos PxQ cuando todos los procesos/máquinas virtuales est'n concentrados en la misma máquina físca, como se puede observar en el caso de resolver problemas dados por una matriz de tamaño 20K x 20K.

Se observa que el hecho de que concentrar la actividad en el mínimo de máquinas físicas juega en contra del rendimiento. Esto es debido a que cada máquina física cuenta con unos

recursos limitados a repartir entre el número de procesos implicados sin contar la memoria o las unidades de procesamiento. Es ahí donde se forma el cuello de botella del sistema y no en las comunicaciones como suele ser habitual.

Por ello, en el caso de que el clúster virtual esté *disperso*, cada máquina virtual única dispone de dichos recursos sin competir con otras máquinas virtuales, lo cual en principio es igual para todas las configuraciones de este escenario y mejor en términos generales. De hecho, en este caso de un clúster *disperso*, el efecto de la rejilla de procesos PxQ es más significativo ya que todas las comunicaciones entre procesos se realizarán de forma externa respecto a las máquinas físicas y tenderá a tener mayor importancia la distribución del problema y las comunicaciones, ya que se tenderá a *preferir* una distribución que implique porciones más cuadradas del problema y por lo tanto una comunicación menos lineal y más orientada a la fluidez de comunicaciones entre los procesos implicados, es decir, será mejor una rejilla de procesos más compensada.

6.6 Resultado Prueba V4

En la prueba V4 se comparan distintos clústers compuestos por un número creciente de máquinas físicas. Cada escenario se ve sometido a un problema proporcional a la memoria total que maneja. Hay dos escenarios virtuales que se comparan en este experimento: una *réplica* virtual del escenario nativo, y el escenario virtual que maximiza el número de máquinas virtuales[35].

Los resultados de la prueba V4 se muestran por medio de las figuras 9 y 10 que muestran las medias de rendimientos obtenidos según el tamaño de bloque NB, medidos en GFLOPS, utilizado para los distintos escenarios en el eje de ordenadas. El eje de abcisas se corresponde con el número de máquinas físicas por las que está compuesto el clúster: 1, 2 o 4 máquinas físicas. Los datos obtenidos por los clústers al utilizar la rejilla PxQ más equilibrada est'an representados en la figura 9, mientras que la figura 10 representa los resultados al utilizar la rejilla PxQ algo descompensada.

Se puede observar que las diferencias entre las figuras 9 y 10 son similares entre sí, especialmente el comportamiento del escenario nativo y la *réplica* virtual.

Si nos fijamos en ambos escenarios parecen ir parejos con una perdida del rendimiento de la *réplica* del 30-50% respecto al escenario natvo, siempre con una ligera mejoría al usar una rejilla de procesos algo descompensada, pero el comportamiento se corresponde con el esperado en cuanto a escalabilidad:

Comparando el escenario nativo con la *réplica* virtual, a medida que el número de máquinas físicas en el clúster aumenta y aunque el problema sea proporcionalmente igual de grande para él, la parte de gestión del clúster aumentan al igual que las comunicaciones para resolver el problema, implicando más competencia para la capa de virtualización en uso de procesamiento y una mayor penalizción para el escenario virtual. Debido a que no se ha continuado aumentando el número de máquinas, es imposible saber a ciencia cierta si esta penalización sigue aumentando o si bien se estabiliza[36].

Distinto es el caso de utilizar el máximo de máquinas virtuales posibles donde si se requieren altos usos de memoria las comunicaciones pesan mucho más en el rendimiento debido a que hay más comunicaciones que son externas a cada máquina virtual en vez de resolverse de forma interna. Dependiendo de cada caso, o bien se necesita capacidad de cómputo de la máquina física para resolverlas de forma interna, o bien se hace un uso más intensivo de la red de interconexión. Esto da en una evolución de rendimientos casi lineal o incluso limitada debido al conjunto del aumento de las comunicaciones y penalizaciones que esto supone en el rendimiento final, agravadas por una rejilla PxQ descompensada.

[35] Tantas máquinas virtuales como unidades de procesamiento tenga el clúster, exceptuando el *frontend*.
[36] Véase los resultados obtenidos en la prueba V3.

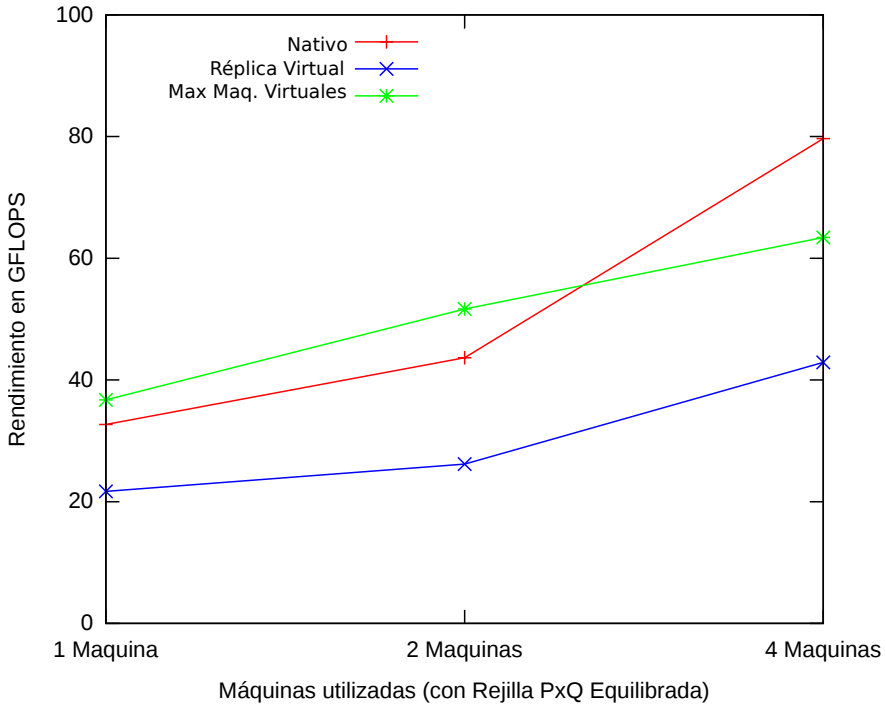

Fig. 9. Medias de rendimientos comparados de clústers compuestos por 1, 2 y 4 máquinas físicas, comparando si dicho clúster se encuentra configurado por medio de máquinas físicas únicamente (Nativo), una unica máquina virtual por cada máquina física (*Réplica Virtual*), o bien utilizando 8 máquinas virtuales por máquina física (Max. Maq. Virtuales). En todos los casos se ha probado un problema que representa el 80% de la memoria disponible y una rejilla de procesos PxQ lo más compensada posible, igual a 2x4 (para el caso de usar 1 máquina física), 4x4 (2 máquinas), o 4x8 (4 máquinas).

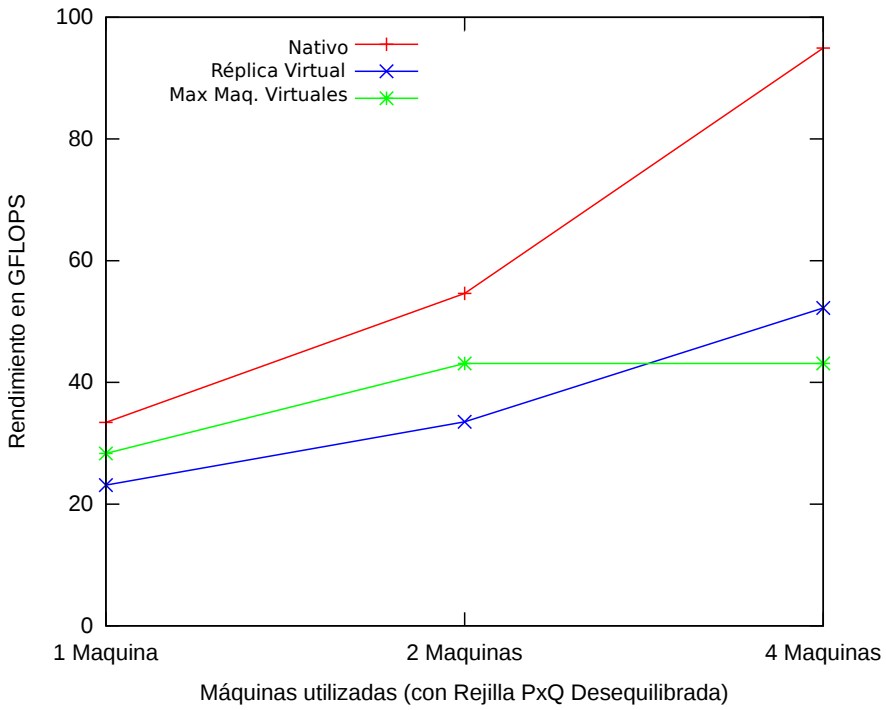

Fig. 10. Medias de rendimientos comparados de clústers compuestos por 1, 2 y 4 máquinas físicas, comparando si dicho clúster se encuentra configurado por medio de máquinas físicas únicamente (Nativo), una unica máquina virtual por cada máquina física (*Réplica Virtual*), o bien utilizando 8 máquinas virtuales por máquina física (Max. Maq. Virtuales). En todos los casos se ha probado un problema que representa el 80% de la memoria disponible y una rejilla de procesos PxQ algo descompensada, lo que es igual a 1x8 (para el caso de hacer uso de 1 máquina física), 2x8 (2 máquinas), o 2x16 (4 máquinas).

No obstante, es curioso como para un número bajo de máquinas utilizadas (una o dos), en caso de utilizar una rejilla de procesos compensada, el clúster virtual con más máquinas virtuales obtiene mejores resultados que el clúster nativo. Esto es achacable a que el clúster *encuentra* para un número relativamente bajo de comunicaciones una forma más fluida de resolverse que no está limitada por la agrupación natural de los recursos en máquinas físicas debido, junto con que este tipo de virtualización ayuda pues *ataca* directamente a los procesadores, y a que el tamaño del problema no es en sí mismo muy grande y por lo tanto no son necesarias demasiadas comunicaciones para resolverlo. Esta misma fluidez se pierde en cuanto las comunicaciones entre procesos se propagan de una forma más lineal, por lo que dicha ventaja se pierde con rejillas de procesos PxQ menos descompensadas, y en cuanto el tamaño del problema aumenta y por lo tanto, proporcionalmente, aumenta el número de comunicaciones necesarias para resolverlo.

7 Conclusiones

Como conclusiones respecto al uso de máquinas virtuales en vez de máquinas físicas en un clúster dedicado a la computación de alto rendimiento, se puede deducir lo siguiente por lo visto en este trabajo:

- El mayor impacto en el rendimiento de un clúster HPC no radica en el uso o no de máquinas virtuales, sino en la configuración de ciertos atributos del clúster relativos al uso de la infraestructura dada. Las características más relevantes[37] son el uso de *HyperThreading* si lo hubiere, la calidad de la red de interconexión interna y la distribución de procesos. Tanto *HyperThreading* como la calidad de la red son factores puramente HW en los que suele existir poca libertad de elección, mientras que la distribución de procesos es un factor SW que es configurado por el usuario/programador, que predispone los flujos de comunicaciones entre procesos.

- Entornos en los que el la interconexión física sea Ethernet 10/100 compensa el uso de máquinas virtuales ya que la interconexión interna de los nodos de computación es un cuello de botella tan significativo que otros factores son irrelevantes.

- El impacto de la virtualización en el rendimiento de un clúster HPC está influido por dos factores, estrechamente relacionados con el uso de la red de interconexión: la distribución de procesos[38] que es un factor inherente a la clústerización, y la configuración y distribución de las VM que, obviamente, está implícito al utilizar una virtualización en una máquina física con varias unidades de procesamiento.

  Al configurar y distribuir las VM en el clúster, se predetermina de alguna forma las comunicaciones entre procesos, y se definen dos características esenciales del escenario con el que trabajar: el número total de nodos y el número de procesadores por nodo. Estas dos características son dos caras de una misma moneda, ya que si tenemos un gran número de nodos, estos no pueden tener un gran número de unidades de procesamiento, y viceversa, si queremos nodos con gran capacidad de cómputo tendremos un escenario con pocos nodos; si minimizamos ambos el número de nodos y el número de unidades de procesamiento por nodo, se estarían infrautilizando los recursos disponibles. Así mismo, al definir el número de máquinas virtuales creadas en cada máquina física y que recursos se asignan a cada máquina virtual se establece una topología de recursos lógicos que predetermina las comunicaciones los nodos y por lo tanto el uso de la red de interconexión.

  La distribución de procesos, independientemente de la virtualización, supone un mapa lógico de los procesos implicado. Si ésto entra en conflicto con la topología de las VM, el rendimiento puede verse afectado. Además, y según los resultados, se recomienda una comunicación asíncrona entre procesos independientemente de la naturaleza y configuración del clúster.

- Los resultados de rendimiento que se obtienen al maximizar el número de máquinas virtuales, sin importar su carga de trabajo, parece ser un indicador del rendimiento de cualquier escenario virtual cuando se llegan a valores límite de problema que el clúster puede resolver. No obstante, maximizando el número de máquinas virtuales no se es capaz de llegar a dichos valores límite.

[37] Obviamente al margen de la potencia de cómputo y cantidad y calidad de memoria principal.
[38] Por medio de la rejilla PxQ en HPL.

– Se ha observado que, en casos en los que el problema a resolver no supere el 50% de la memoria principal, parece indicado el uso de una *réplica* virtual basada en contenedores, ya que es el único caso en el que haciendo uso de la virtualización se obtienen mejores rendimientos que de forma nativo.

– A medida que el tamaño del problema supera el 50% la memoria del sistema y se acerca al límite de la misma se reducen las diferencias de rendimiento entre escenarios virtuales y aumenta la diferencia respecto al escenario nativo. En tales casos, podría compensar el uso de un escenario virtual con más máquinas virtuales que físicas si, a pesar de la pérdida de rendimiento, se busca un escenario con más nodos.

Si se utilizan configuraciones con varias VM por máquina virtual y varias unidades de procesamiento por VM se tiende a sobrecargar tanto la capacidad de cómputo dedicada a la virtualización como las comunicaciones nivel interno dentro de cada máquina física debido a los distintos procesos que representan las distintas máquinas virtuales. Esto no compensa para tamaños de problema pequeños/medios pero sí puede compensar para tamaños de problema medios/grandes. Por lo que una topología de procesos adecuada puede ser importante en el sentido de que ayude a fluir las comunicaciones propias del problema a resolver.

Como observación final, y aunque no se haya visto reflejado este aspecto en las pruebas y resultados, en cuestiones de aislamiento de sistemas y procesos no es recomendable desde el punto de vista del autor realizar trasiegos, reutilizaciones, o apagados y encendidos de los clústers virtuales, ya que el sistema utilizado tiende a no liberar correctamente los recursos asignados a los procesos que representan las máquinas virtuales y se producen fallos permanentes en el clúster virtual imposibles de ignorar y/o solucionar. Es preferible seguir un esquema de Creación/Configuración del clúster virtual - Utilización del clúster virtual - Eliminación del clúster virtual, lo cual hay que tener en cuenta si se quiere utilizar el cluster como *IaaS* o *PaaS*.

8 Trabajo Futuro

En un futuro se puede ampliar este estudio en varios puntos, como un incremento en el número de máquinas físicas utilizadas, una actualización de los elementos (HW o SW) utilizados, o un cambio del paradigma de virtualización utilizado.

En primer lugar, si atendemos a las necesidades de estructura HW de un clúster HPC, sería deseable poder probar los escenarios con un mayor número de máquinas físicas con el fin de comprobar el comportamiento en relación a la escalabilidad. Asimismo también sería deseable probar el clúster en un entorno con un sistema de interconexión típico de un escenario HPC, como puede ser una interconexión Infiniband, de forma que reduzca el cuello de botella que suponen las comunicaciones internas en un clúster, en la medida de lo posible.

En segundo lugar, si atendemos a la infraestructura SW de un entorno HPC, es justo sustituir HPL por HPCC como *benchmark* a utilizar con el fin de obtener más y mejor información de los comportamientos en cuanto a uso de memoria y ancho de banda de la red de interconexión interna. No obstante, se recuerda que HPL es una de las 7 subpruebas de HPCC, así pues, las experiencias recogidas en este estudio no resultan en balde. De forma complementaria a este cambio de *benchmark* puede ser necesario un cambio en las bibliotecas matemáticas utilizadas, cambiando la ya obsoleta y descontinuada GotoBLAS2 por su sucesora OpenBLAS (o por cualquier otra biblioteca tipo *BLAS* que esté en vigor). Por otra parte SGE (ahora Oracle Grid Engine) no es la única forma de gestionar los recursos y tareas en un clúster, hay alternativas como TORQUE o SLURM que están ganando popularidad y se invita a utilizar.

Y por último, pero quizás sea el aspecto más importante en cuanto a trabajo futuro, probar otras técnicas de virtualización, especialmente utilizando un *hypervisor* de Tipo II como es Xen si se quiere explotar el máximo rendimiento, o bien de Tipo I si se quiere ganar en flexibilidad (como puede ser VMWare o VirtualBox) aunque se prevé una pérdida de rendimiento considerable. También se pueden probar otras técnicas de virtualización basada en contenedores como LxC o Docker, por ejemplo.

9 Bibliografia

References

1. "Research on the application of virtualization technology in high performance computing". Yan Junhao and Lv Aili; Henan Polytechnic University, Jiaozuo, China. In proceeding of: Electrical and Electronics Engineering (EEESYM), 2012 IEEE Symposium on. 2012
2. "System-Level Virtualization for High Performance Computing". Geoffroy Vallée , Christian Engelmann , Stephen L. Scott , Thomas Naughton , Hong Ong ; Oak Ridge National Laboratory, Tennesse, USA. 16th Euromicro Conference on Parallel, Distributed and Network-Based Processing, pp 636-643. Feb 2008
3. "Virtualized HPC: a contradiction in terms?". Birkenheuer, Georg and Brinkmann, André and Kaiser, Jürgen and Keller, Axel and Keller, Matthias and Kleineweber, Christoph and Konersmann, Christoph and Niehörster, Oliver and Schäfer, Thorsten and Simon, Jens and Wilhelm, Maximilian. published in "Software: Practice and Experience, year 2012 vol 42, number 4", pp 485-500. 2012
4. HPL - A Portable Implementation of the High-Performance Linpack Benchmark for Distributed-Memory Computers, `http://www.netlib.org/benchmark/hpl/`
5. HPL ICL-UT (Innovative Computing Laboratory, Universiy of Tennessee), `http://icl.cs.utk.edu/hpl/`
6. "Toward a New Metric for Ranking High Performance Computing Systems" Heroux, M. A., Dongarra, J. UTK EECS Tech Report and Sandia National Labs Report SAND2013-4744, June 2013. `http://icl.cs.utk.edu/news_pub/submissions/HPCG-Benchmark-utk.pdf`
7. "Performance of Various Computers Using Standard Linear Equations Software, (Linpack Benchmark Report)". Jack J. Dongarra, University of Tennessee. Published in "Computer Science Technical Report" CS-89-85, 2013. `http://www.netlib.org/benchmark/performance.ps`
8. "The LINPACK Benchmark: Past, Present, and Future" Dongarra, J., Luszczek, P., Petitet, A. Published in "Concurrency: Practice and Experience, 15" pp. 803-820, 2003. `http://www.netlib.org/utk/people/JackDongarra/PAPERS/hplpaper.pdf`
9. Rocks Clusters Official Website, `http://www.rocksclusters.org/wordpress/`
10. HPL Generator `http://www.hpcgeeks.com/index.php/hpc-benchmarking/linpack/6-hpl-dat-file-generator`
11. "The performance impact of computational efficiency on HPC clústers with Hyper-Threading technology" O. Celebioglu, A. Saify, T. Leng, J. Hsieh, V. Mashayekhi, R. Rooholamini. Published in "Parallel and Distributed Processing Symposium International , vol. 15", p. 250b, 2004
12. "An Empirical Study of Hyper-Threading in High Performance Computing clústers". T. Leng, R. Ali, V. Mashayekhi, J. Hsieh and R.Rooholamini. Published in "The Third LCI International Conference on Linux clústers: The HPC Revolution 2002", October 2002.
13. "Hyper-Threading may be Killing your Parallel Performance" by Dr Donald Kinghorn, Posted on July 2, 2014 `http://www.pugetsystems.com/blog/2014/07/02/Hyper-Threading-may-be-Killing-your-Parallel-Performance-578`
14. "El libro del administrador de Debian", 12.2. Virtualización `http://debian-handbook.info/browse/es-ES/stable/sect.virtualization.html`
15. "An Updated Performance Comparison of Virtual Machines and Linux Containers" Wes Felter, Alexandre Ferreira, Ram Rajamony, Juan Rubio. Published in "Technology, 28, 32" (2014)
16. "Performance evaluation of container-based virtualization for high performance computing environments." M. Xavier, M. Neves, F. Rossi, T. Ferreto, T. Lange, and C. De Rose. Published in "Parallel, Distributed and Network-Based Processing (PDP), 21st Euromicro International Conference on pages 233–240", Feb 2013.
17. "Performance Comparison of Hardware Virtualization Platforms" Daniel Schlosser, Michael Duelli and Sebastian Goll. University of Wüzburg, Institute of Computer Science. Published in "NETWORKING 2011: 10th International IFIP TC 6 Networking Conference, Valencia, Spain, May 9-13, 2011, Proceedings" pp 393-405. 2011

A HPL

HPL[4] son las siglas que se corresponden a *High Performance Linpack*. Linpack fue desarrollado por Jack Dongarra en 1976 y fue pensado como un *benchmark* destinado a medir el rendimiento de sistemas informáticos por medio de la biblioteca Linpack. Pese a que Linpack se está viendo sustituido por la biblioteca Laplack (la cual es mas eficiente en sistemas de arquitectura RISC), y pese a las críticas que pesan sobre Linpack, todavía Linpack resiste como una medida de referencia debido a su capacidad de ampliación, y a que da un número único de rendimiento (en comparación con otros *benchmark*).

Hay tres versiones históricas del *benchmark* Linpack: Linpack 100 (tamaño de la matriz 100x100), Linpack 1000 (tamaño de la matriz 1000x1000) y HPL. Las dos primeras no son adecuadas para entornos paralelos ni son escalables en tamaño, motivo por el cual surgió HPL, donde la matriz a resolver se puede hacer tan grande como sea necesario. En este estudio se utiliza la variante para computación de alto rendimiento HPL.

Pero ¿En qué consiste Linpack? Linpack es un *benchmark* que resuelve la factorización `LU` (con pivoteo parcial) de un sistema *denso*[39] de ecuaciones lineales `Ax = b` generado al azar. Asimismo los elementos de la matriz son de tipo coma flotante de doble precision (64 bits) en un sistema de memoria distibuida.

El algoritmo usado por HPL puede resumirse en los siguientes conceptos:

1. Distribución ciclica de datos en bloques bidimensionales.
2. Factorización LU con pivoteo parcial de fila con múltiples profundidades de anticipación.
3. Factorización recursiva de los paneles con la búsqueda de pivote y difusión de columna combinados.
4. Intecambio y difusión (*swap/broadcast* de los resultados provisionales.

El paquete de HPL proporciona un programa que realiza el *testeado* y *timing* para cuantificar la precisión de la solución obtenida, así como el tiempo que tomó para calcularla. El mejor rendimiento alcanzable por este software en su sistema depende de una gran variedad de factores (como por ejemplo la red de interconexión utilizada). Sin embargo, HPL es escalable en el sentido de que su eficiencia se mantiene constante en paralelo con respecto al uso de la memoria por procesador.

HPL contiene muchas variantes de ejecución ya que así se decidió cuando se diseñó HPL: dejar la oportunidad de configuración de ciertos parámetros a disposición del usuario. Por esto, debido a las múltiples configuraciones/posibilidades/requisitos de las máquinas, no hay una única posibilidad de probar HPL pero sí nos aseguran que desde el punto de vista de precisión numérica, todas las combinaciones posibles son rigurosamente equivalentes.

Algoritmo Principal: Como ya hemos mencionado, HPL resuelve un sistema de ecuaciones lineales de orden $\boldsymbol{n}$: `Ax = b` por factorización `LU` con pivoteo parcial de filas de la matriz de elementos del sistema: `[A b] = [[L,U] y]`. Es desde la matriz inferior `L`, la cual se mantiene sin pivotar, desde la cual se va aplicando a `b` los progresos de la factorización. La solución `x` es obtenida por medio de la resolución del sistema `Ux = y`. Al finalizar no se devuelve el array de pivotes.

La distribución de los datos se realiza en una rejilla de procesos bidimensional `PxQ`, acorde al esquema cíclico de bloques, para asegurar un buen balanceo de carga y escalabilidad del algoritmo.

[39] La matriz a resolver tiene mayoria de elementos que son distintos de 0, a diferencia de una matriz *sparse*.

Tal y como se muestra en 11, la matriz de elementos ***N-por-(N+1)*** es dividida en bloques de tamaño ***NB-por-NB*** índices[40] que son asignados ciciclamente a los procesos de la rejilla `PxQ`: Asigna la primera fila de bloques de tamaño *NB-por-NB* a la primera fila de procesos (dichos bloques son distribuidos de forma que el primer bloque *NB-por-NB* va al proceso P_1xQ_1, el segundo bloque al proceso P_1xQ_2, ... el *i-ésimo* al proceso P_1xQ_q que es el último de la rejilla, el siguiente bloque *i+1-ésimo* será asignado al proceso P_1xQ_1, el bloque *i+1-ésimo* al proceso P_1xQ_2, ...) , la segunda fila de bloques a la segunda fila de procesos (el primer bloque *NB-por-NB* va al proceso P_2xQ_1, el segundo bloque al proceso P_2xQ_2, ...) , ... la i fila de bloques es asignada a la i *Módulo P*. De tal forma que, debido a la forma en la que está distribuida la memoria[41] la i fila de bloques es asignada a la fila de procesos i *Mod* P; dentro de una fila de procesos j, la j columna de datos (de tamaño $NB\ x\ NB$) será asignada al proceso $P_{(jmodP)}xQ_i$

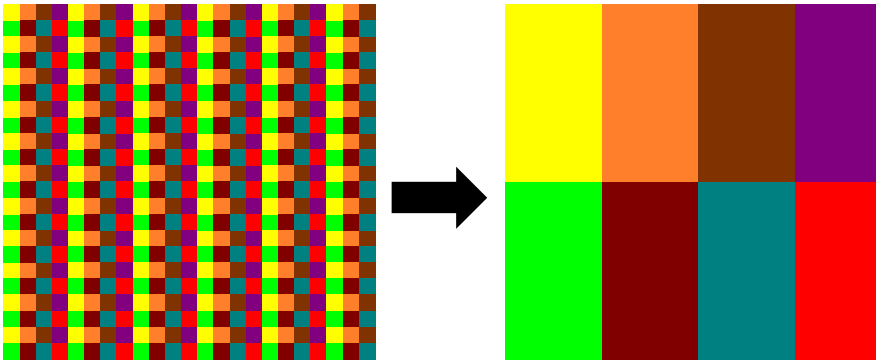

Fig. 11. Como se ve, primero la matriz de elementos es dividida en bloques (de tamaño *n-por-(n+1)* los cuales son asignados a los procesos agrupados en la rejilla *PxQ* de una forma ciclica: Asigna la primera fila de bloques a la primera fila de procesos, la segunda fila de bloques a la segunda fila de procesos, ...

Aunque se propone la variante de la búsqueda por la derecha del bucle principal de la factorización `LU`[42], lo importante sin importar la variante escogida, es que cada iteración del bucle un panel de **NB** columnas es factorizado y la submatriz de salida es actualzada.

Factorización del Panel: En cada iteración del bucle principal, cada factrización del panel se produce en una columna de procesos, como se observa en la figura 11. Esta parte de la asignación de parcelas de la matriz a los procesos es considerada crítica.

HPL permite al usuario elegir las variantes recursivas a ejecutar de factorización LU, en cuantos subpaneles puede ser divido un panel de forma recursiva, imponer una limitación por tiempo a la factorización del subpanel, o una combinación de las anterios.

Difusión del Panel: Una vez que un panel ha sido factorizado, este panel se distribuye a los demás procesos en base a uno de los seis posibles metodos de difusión: ***Increasing-ring***, ***Increasing-ring (modificado)***, ***Increasing-2-ring***, ***Increasing-2-ring (modificado)***, ***Long (Reducción de ancho de banda***, y ***Long (Reducción de ancho de banda, modificado***.

Las variantes *Ring* se basan en clásicos esquemas más o menos lineales de comunicación síncrona. Las variantes *Long* son de comunicación síncrona entre todos los procesos implicados (todos los Q_j de un mismo P_i), de forma que mas mensajes son intercambiados[43] haciendo de este algoritmo especialmente adecuado para mensajes largos.

[40] Dichos tamaños a su vez son los utilizados para la distribución de datos.
[41] De forma configua
[42] Las alternativas a dicha variante son busqueda por la izquerda y por el método de Crout.
[43] El volumen total de comunicaciones es independiente de Q

Previsión *look-ahead*: Una vez que el panel ha sido difundido (o mientras se produce la difusión), la submatriz de salida se actualiza con el último panel en el *pipe* de *look-ahead*[44]. Este paquete permite seleccionar la profundidad de varias previsiones (*look-ahead*). Por convenio, una profundidad igual a 0 corresponde a ninguna de búsqueda hacia delante, en cuyo caso la submatriz de salida se actualiza por el panel que está siendo difundido en ese momento. *Look-ahead* consume algo de memoria extra para mantener actualizados todos los paneles de columnas en el *pipe* de previsiones. Con una *previsión/look-ahead* de profundidad 1 (tal vez 2) se suelen obtener mejores rendimientos.

Actualización: La actualizacion de la submatriz de salida por el ultimo panel en la tubería de *look-ahead* se realiza en dos fases:

1. Ordenación de los pivotes para formar la actualización del correspondiente panel fila `U`.
2. Difusión de `U`a cada fila de procesos para que la actualización de rango `NB` pueda realizarse.

Sustitución hacia atrás: Cuando la factorización ha terminado se realiza la sustitución hacia atrás, para lo cual se elige un *look-ahead* de profundidad 1. El lado derecho[45] es enviado a las filas del proceso[46], de forma que se reseuelven `Q*NB` entradas al mismo tiempo. Por cada paso estsegmento cada vez más reducido del lado derecho[47] se actualiza. El proceso situado encima del que posee el actual bloque diagonal de la matriz `A` actualiza primero sus ultimos `NB` segmentos de `x`, se los envía al proceso de la columna anterior, y luego difunde entre la columna de procesos siguiendo la configuración de distribución acordada. La solucion de cada sistema lineal se deja copiada en cada proceso fila.

Chequeo de la solución: Para verificar el resultado obtenido, la matriz de entrada y lado derecho se regeneran. Tres residuales, mostrados en la figura 12, se calculan y una solución se considera como *numéricamente correcta* si todas estas cantidades son inferiores a un valor de umbral del orden de 1. En las expresiones de abajo, `eps` es un factor relativo precisión de la máquina. Si se supera dicho valor umbral, se consiedarará que esa parte de la prueba no ha sido superada.

```
||Ax-b||_oo / ( eps * ||A||_1 * N )
||Ax-b||_oo / ( eps * ||A||_1 * ||x||_1 )
||Ax-b||_oo / ( eps * ||A||_oo * ||x||_oo )
```

Fig. 12. Cálculo de los residuals

HPL ¿Cómo instalarlo? En el Apéndice A se adjunta el código de un script para bash que automatiza la instalación de HPL en un entorno Rocks Clusters en sus versiones 6.1 y 6.1.1. Puede servir de referencia para otros entornos (con especial atención a la sustitución en el fichero `Make`file correspondiente del paqute de software `GotoBLAS2`[48].[49] .

[44] La factorización del panel se encuentra en la ruta crítica, lo que significa que cuando el panel de orden k ha sido factorizado y difundido, la próxima tarea más urgente para completar es la factorización y de difusión del panel $k+1$. Esta técnica se refiere a menudo como *"look-ahead"*, lo cual podría traducurse por *previsión*.

[45] En la variante de la búsqueda por la derecha.

[46] Según como se haya seleccionado la forma de difusión de los paneles.

[47] De nuevo dependiendo de que hayamos escogido esta opción.

[48] **NOTA IMPORTANTE:** Este software es necesario y se tiene que bajar con un navegador web. No se puede bajar por consola, por lo que el script adjunto no descarga autom'aticamente dicho software.

[49] Como ya hemos mencionado, se puede utilizar OpenBLAS o GotoBLAS2. Son prácticamente iguales en cuanto a su instalación y uso.

HPL.dat `HPL.dat` es el fichero de configuración que lee el programa `xhpl`[50] y que almacena todas las opciones, variantes tamaños (de bloque, de problema, de rejilla, ...) con las que ejecutar `HPL`.

El fichero `HPL.dat` debe ser obligatoriamente llamado así así que por cada configuración distinta a probar debemos esperar a que termine la prueba correspondiente, cambiar los valores de `HPL.data` lo que representen la configuración deseada y correr `xhpl`; esperar a que acabe, y de nuevo cambiar los valores internos de `HPL.dat`, aunque queramos lanzar dos pruebas en dos conjuntos distintos de ordenadores, no podremos si están gestionadas en un mismo cluster.

En el Ápendice B mostraremos un ejemplo de `HPL.dat` utilizado para correr las pruebas, y lo utilizaremos a su vez para explicar que se puede configurar con dicho fichero (y los valores que hemos seleccionado)[51].

Las lineas 1 y 2 son ignoradas y sirven para hacer algún comentario. Las lineas que controlan los parámetros más relevantes son (por orden de aparición en el fichero):

- La línea 3 es donde decimos en que fichero queremos que nos guarde la salida.
- La línea 4 es para indicar que tipo de salida queremos (8 es para el fichero, 6 y 7 para que nos de la salida por pantalla o por la salida de error standard respectivamente).
- En la fila 5 indicamos cuantos tamaños **N** de problema queremos probar, y en la fila 6 los tamaños **N** propiamente dichos a probar. El tamaño adecuado de **N** se calcula pensando en que la matriz de elementos del problema tiene tamaño **N**x**N** y que cada coeficiente ocupa 64 bits (8 Bytes), por lo que el tamaño de una matriz de elementos de 8 bytes de tamaño **N**x**N** debe ajustarse lo mejor posible a la suma de las memorias de los nodos implicados. Por lo que si tenemos 4 nodos de 8GB[52], una matriz de lado N es la que se adapta al máximo de la memoria de nuesto cluster[53], con N:

$$N = \sqrt{\frac{4nodos * 8GB/nodo * 1024MB/GB * 1024KB/MB * 1024B/KB}{8B/Elemento}}$$

  Así pues nuestro máximo tamaño **N** al que deberíamos someter nuestro sistema es de 65536 (y el 80% ronda los 52428 elementos de lado). En nuestro caso probaremos con los tamaños de lado 10000, 20000, 40000, 50000, 60000, y 63498 (por abarcar una horquilla de tamaños desde lo mínimo para un problema razonable, hasta acercarnos al límite máximo calculado con el fin de observar tambieén la escalabilidad del tamaño de problema en nuestro sistema).
- La línea 7 indica cuantos tamaños de bloque **NB** vamos a probar, y la línea 8 indica los tamaños de bloque **NB** a probar. Este paramétro es de los más sensibles para la ejecución de HPL (ya que de él depende la sobrecarga de las comunicaciones y el tamaño de los paneles). Se recomienda un NB no superior a 256 y no inferior a 32, de forma que etablecemos una horquilla que recorra esos valores con el fin de averiguar que NB se adapta mejor a nuestro sistema.
- Las lineas 10, 11 y 12 indican, respectivamente, cuantas rejillas `PxQ`, y las correspondientes `P` (línea 11, un `P`por columna) y `Q` (línea 12, un `Q`por columna). De forma que la misma columna i (con i entre 0 y el número indicado en la línea 10) de las filas 11 y 12 conforman una rejilla `PxQ` a probar. Nosotros probamos con rejillas tales que `Q` sea mayor o igual que `P` [54] y a su vez:

[50] El programa que realmente corre el *benchmark*.

[51] Realmente hemos generado el HPL.dat por medio de la aplicación web `http://www.hpcgeeks.com/index.php/hpc-benchmarking/linpack/6-hpl-dat-file-generator`. Nos hemos limitado a modificar los parmétros que hemos considerado necesarios.

[52] No obstante, según la fuente empleada para obtener la memoria nos da un valor de 7960MB (con *free -mt* o de 7'5GB (con las herramientas propias de *Rocks*

[53] No obstante recomiendan ajustar el valor de N al 80% del máximo dado por la memoria de nuestro cluster.

[54] Se recomienda que P y Q estén equilibrados, pero que en caso de desequilibrio (porque no se pueda hacer un PxQ cuadrado perfecto, o porque incluso un poco de desequilibrio puede ser beneficioso, además que justo nuestro cluster no permite grandes desequilibrios) que Q sea ligeramente a P

$P * Q$ = **Total Cores Sistema** = (Número de Nodos) x (Cores por Nodo)[55]

Los parámetros más importantes (por lo sensible que es HPL ante estos aspectos antes que a otros) son ya comentados: el/los tamaño de lado de matriz de elementos **N**, el/los tamaño/sde bloque **NB**, y la/s configuracion/es de rejilla **PxQ**. A partir de aquí, dejamos el fichero tal y como lo obtenemos de [10] según la configuración de nuestro sistema. El resto básicamentr trata otros aspectos de configuración ya comentados:

- La línea 9 especifica como deben ser mapeados los procesos MPI en los nodos de nuestro sistema. Pueden ser por fila o por columna principal. Se recomienda por fila principal.
- En la línea 13 se espeficica el umbral de error de los *residuals* nombrados en la seccion de chequeo en el algoritmo. Pese a que es un valor importante y su ajuste puede dar mucho juego, se recomienda 16 como valor adecuado para la mayoria de los casos (que es como lo tenemos).
- Las lineas de la 14 a la 21 tratan acerca de las opciones de configuración del algoritmo (siempre se indica primero el número de configuraciones a probar del parámetro en cuestión y en la línea siguiente los valores de las tantas configuraciones indicadas en la primera linea): variante de LU utilizada para factorizar el panel (búsqueda por la derecha, por la izquerda o Crout), criterio de parada de recursividad (tamaño de bloque mínimo), número de subpaneles a dividir cada panel en cada recursión y método de factorización en cada subpanel (de nuevo búsqueda por la derecha, por la izquerda o Crout). En nuestro caso viene configurado para que solo utilicemos una única configuración con variante búsqueda por la derecha en el panel, divisiones en 2 paneles por cada recursión (mientras que el tamaõ sea superior a 4 índices), y factorizacion por Crout para el subpanel.
- Las lineas 22 y 23 indican el número de tipos de difusión a utilizar y el/los tipos de difusión a utilizar (anillo 1, anillo 2, long, o una modificación de las nombradas). En nuestro caso solo utilizamos la configuración por *anillo 1* (pese a que quizás nos convendría utilizar una configuración tipo *long*, como veremos).
- Las lineas 24 y 25 indican el número de profundidades a utilizar en las previsiones (*look-ahead*) a utilizar en la prueba y el/los propios niveles de profundidad de *look-ahead* a utilizar. En nuestro caso utilizamos una sola configuracion de nivel de profundidad a nivel 1.
- La línea 26 indica el tipo de intercambio/difusión a emlpear (binary-exchange, long, o mezcla de ambas). La línea 27 indica el umbral de *swap* para el método *binary-exchange*. Se utilizará una mezcla de ambos con un umbral de 60.
- Las lineas 28 y 29 indican si se desea guardar de forma transpuesta (o no, por defecto es que sí) las matrices `L1` y `U` respectivamente.
- La línea 30 activa/desactiva la fase de equilibrado (recomienda no activarla si utilizamos en la línea 26 las opciones *long* o *mix*. Nosotros lo tenemos activado pese dicha recomendación).
- La línea 31 especifica el alineamiento de la memoria. En máquinas modernas puede ser adecueado un valor de 4, 8 o 16, pero un valor inadecuado puede implicar un gran desperdicio de memoria. Viene configurado con valor de 8 y así lo mantendremos.

[55] **NOTA IMPORTANTE**: Estamos utilizando micropocesadores Intel con tecnología *HypetThreading*, esto es que existe una réplica de gran parte de los registros que necesita cada core (o CPU) y, si se activa el *HypetThreading*, el microprocesador "engaña" al sistema operativo para que este tenga la ilusión de que cuenta con el doble de cores (o CPUs) que existen realmente. Es por esto por lo que realizaremos pruebas en las que *PxQ* multipliquen 32 o 16 en función de si tenemos *HyperThreading* activado o no.

B Script de instalación de HPL

Este es un código que automatiza la instlalación de HPL en un entorno Rocks 6.1/6.1.1 . Puede servir de referencia para otros entornos.

```
#!/bin/bash

## Hay que tener bajado el fichero relativo a GotoBLAS2, el cual hay
## que bajar desde https://www.tacc.utexas.edu/tacc-projects/gotoblas2
## (no se puede hacer con wget)

## Bajar el codigo de HPL de http://icl.cs.utk.edu/hpl/software/index
## .html, el cual esta actualizado a HPL v2-1
#   wget http://icl.cs.utk.edu/projectsfiles/hpl/hpl-2.1.tar.gz
#   wget http://www.netlib.org/benchmark/hpl/hpl-2.1.tar.gz

# Instalar primero GotoBLAS v2.13
tar -xvf GotoBLAS2-1.13.tar.gz
cd /export/home/test/GotoBLAS2

#/export/home/test/GotoBLAS2/quickbuild.64bit
# Si da problemas la l\'inea anterior, comentarla
# En cuanto a la instalacion en maquinas virtuales, en la linea
# de abajo hay que quitar  BINARY=64
make TARGET=NEHALEM BINARY=64
cd

# Instalar ahora el propio HPL v2.1
tar -xvf hpl-2.1.tar.gz
ln -s /export/home/test/hpl-2.1 /export/home/test/hpl

# Sustituimos ciertas librerias y directorios donde Rocks tiene
# dichos recursos
cat /export/home/test/hpl/setup/Make.Linux_PII_FBLAS | \
sed s/'TOPdir       = $(HOME)\/hpl' /
     'TOPdir       = \/export\/home\/test\/hpl'/g |
    sed s/'MPdir        = \/usr\/local\/mpi' /
     'MPdir        = \/opt\/openmpi'/g |
    sed s/'MPlib        = $(MPdir)\/lib\/libmpich.a' /
     'MPlib        = $(MPdir)\/lib\/libmpi.a'/g |
    sed s/'LAdir        = $(HOME)\/netlib\/ARCHIVES\/Linux_PII' /
     'LAdir        = \/export\/home\/test\/GotoBLAS2'/g |
    sed s/'LAlib        = $(LAdir)\/libf77blas.a $(LAdir)\/libatlas.a'
     /'LAlib        = $(LAdir)\/libgoto2.a -lm'
     '-L\/usr\/lib\/gcc\/x86_64-redhat-linux\/4.4.7'/g |
    sed s/'CC           = \/usr\/bin\/gcc' /
     'CC           = \/opt\/openmpi\/bin\/mpicc'/g |
    sed s/'LINKER       = \/usr\/bin\/g77' /
     'LINKER       = \/opt\/openmpi\/bin\/mpif77'/g \
    > /export/home/test/hpl/Make.Linux_PII_FBLAS

cd hpl
make arch=Linux_PII_FBLAS
rocks list host | grep compute | awk '{print $1}' | sed s/':'//g  \
> /export/home/test/machines
##Guardamos el HPL.dat original ya que jugaremos a correr el benchmark
## modificandolo
cp  /export/home/test/hpl/bin/Linux_PII_FBLAS/HPL.dat \
/export/home/test/hpl/bin/Linux_PII_FBLAS/HPL.dat.original

cd
###Para correr el benchmark
#cd /export/home/test/hpl/bin/Linux_PII_FBLAS/
#/opt/openmpi/bin/mpirun -nolocal -np 2 \
#-machinefile /export/home/test/machines xhpl
```

C Ejemplo de *HPL.dat*

Este es un ejemplo de un fichero de configuración de HPL el cual siempre es el mismo para cualquier ejecución y se llama HPL.dat.

```
HPLinpack benchmark input file
Innovative Computing Laboratory, University of Tennessee
FU-SinHT-0032to0256.out      output file name (if any)
8            device out (6=stdout,7=stderr,file)
6            # of problems sizes (N)
10000 20000 40000 50000 60000 63498 Ns
9           #NBs
32 44 64 96 128 132 144 168 256 NBs
0            PMAP process mapping (0=Row-,1=Column-major)
3            # of process grids (P x Q)
1 2 4        Ps
16 8 4       Qs
16.0         threshold
1            # of panel fact
2            PFACTs (0=left, 1=Crout, 2=Right)
1            # of recursive stopping criterium
4            NBMINs (>= 1)
1            # of panels in recursion
2            NDIVs
1            # of recursive panel fact.
1            RFACTs (0=left, 1=Crout, 2=Right)
1            # of broadcast
1            BCASTs (0=1rg,1=1rM,2=2rg,3=2rM,4=Lng,5=LnM)
1            # of lookahead depth
1            DEPTHs (>=0)
2            SWAP (0=bin-exch,1=long,2=mix)
64           swapping threshold
0            L1 in (0=transposed,1=no-transposed) form
0            U  in (0=transposed,1=no-transposed) form
1            Equilibration (0=no,1=yes)
8            memory alignment in double (> 0)
```

D Rocks Cluster

Este anexo recoge a modo de tutorial, la forma de uso de Rocks Cluster 6.1/6.1.1 para crear cluster en su forma nativa o virtual. Toda la información aquí expuesta se puede encontrar en `http://central6.rocksclusters.org/roll-documentation/base/6.1.1/` *en caso de tratarse de un clúster nativo, y* `http://central6.rocksclusters.org/roll-documentation/kvm/6.1.1/` *para ampliar la información en caso de tratarse de un clúster virtual. Se dará por sentado que se dispone de una serie de equipos, conectados entre sí por medio de una red de interconexión interna, y uno de los equipos ya instalado como* frontend *(equipo que utilizaremos para gestionar, controlar y manejar todo el cluster). Si se va a crear o se tiene intención de crear un cluster virtual es* MUY IMPORTANTE *instalar el* roll KVM *durante la instalación del* frontend*, no obstante, se dejará todo lo relativo a clústers virtuales para un subapartado de este anexo.*

Preparación del Clúster: *Antes de nada, se debe permitir el acceso al cluster desde el exterior (si se desea):*

```
$ rocks remove firewall host=localhost rulename=A40-WWW-PUBLIC-LAN
$ rocks add firewall host=localhost network=public protocol=tcp \
     service=www chain=INPUT action=ACCEPT flags="-m state --state NEW \
     --source 0.0.0.0/0.0.0.0" rulename=A40-WWW-PUBLIC-NEW
$ rocks sync host firewall localhost
```

Y crear un usuario alternativo a root *que es el que se utilizará para manejar el clúster:*

```
$ useradd USUARIO
$ passwd USUARIO
$ rocks sync users
```

Sería deseable tener apuntados las direcciones MAC que tienen el resto de equipos (los que van a realizar tareas de nodo o de contenedor), ya que las tareas de encendido se pueden realizar desde el frontend *siempre y cuando el equipo a encender tenga habilitado en BIOS esta opción. En cualquier caso, los equipos deben tener habilitada la opcion* PxE *tal y como se indica en la documentación oficial de Rocks Cluster. Para instalar los nodos o contenedores que van a componer el clúster, desde el* frontend*, con cuenta de* root *hay que teclear el comando:*

```
$ insert-ethers
```

*Seleccionar la opción que corresponda (*Compute *o* VM Container*) y encender los equipos que se deseen. Si se conoce la dirección MAC del equipo, y está habilitado en el mismo la opción de arrancado por red, entonces se puede teclear el siguiente comando desde el frontend (en una terminal distinta a la que se utiliza para el comando* insert-ethers*) para encender los equipos.*

```
$ ether-wake -i eth0 [DIRECCION MAC]
```

Cada uno de los equipos a instalarse en el clúster irá recibiendo un nombre como compute-X-Y *o* vm-container-X-Y*, según corresponda su naturaleza, y con los índices X e Y en orden consecutivos (en principio asignados automáticamente). Una vez que, en la pantalla de* insert-ethers*, todos los los equipos a instalar son marcados con un asterisco se puede proceder a salir de la pantalla de* insert-ethers*. Se puede monitorizar la progresión de las instalaciones con el siguiente comando, que no deja de ser un cliente de VNC*[56]*:*

```
$ rocks-console compute-X-Y
```

Llegados a este punto, ya deberíamos tener un clúster compuesto por un frontend *y una serie de nodos destinados a la computación o a alojar máquinas virtuales.*

[56] IMPORTANTE: Si se está trabajando a distancia, para poder utilizar este comando, habrá que acceder al *frontend* mediante *ssh* con la opción *-X* y así permitir la ejecución del entorno gráfico.

Manejo del Clúster: *Supongamos que tenemos un clúster destinado a computación (es decir, los nodos son nodos de computación), para poder manejar el clúster es necesario realizar varias comprobaciones y acciones relativas al gestor de tareas. El gestor de tareas (SGE por defecto en Rocks Cluster) coordina equipos/grupos de equipos (hostgroups) agrupados en colas de trabajo (queue), para realizar determinadas tareas por medio de distintas tecnicas de programación paralela (entornos paralelos). Pese a que por defecto se configurarán y se ofrecerá al usuario al menos una cola de trabajo (*all.q*), un grupo de equipos (*@allhosts*) y un típico entorno paralelo para trabajar con MPI (mpi, mpich, orte), se recomienda en cada caso no trabajar directamente con cada uno de ellos, sino replicar cada uno de dichos elementos a utilizar y ya trabajar sobre ellos. Cada uno de estos elementos viene dado por una configuración que está en un fichero de texto plano independiente, por lo que se puede copiar su contenido y editarlo fácilmente, aunque se recomienda utilizar siempre que se pueda la herramienta* qconf. *Recuerdese que se está trabajando por defecto con la cuenta de* root*, pero si se desea se pueden otorgar permisos de manejo y administración del gestor de tareas por medio del siguiente comando:*

```
$ qconf -am USUARIO
$ rocks sync users
$ rocks sync config
```

Empezaremos por el elemento relativo a los entornos paralelos ya que no depende ni de las colas de trabajo, ni de los grupos de equipos. Primero se consultan los entornos paralelos disponibles con este comando

```
$ qconf -spl
```

El típico entorno paralelo es mpi*, por lo que lo que se le tomará como ejemplo. Como se ha dicho, se debe evitar trabajar directamente con dicho entorno ya preconfigurado debido a que a veces la configuración predeterminada del entorno no es adecuada con el programa que se quiere ejecutar, así que se copia el contenido de la configuración del entorno paralelo en otro fichero:*

```
$ qconf -sp mpi > /nuestra/ruta/ejemplo/a/MyMPI.txt
```

Hay un campo llamado pe_name *que se debe modificar para que no coincida con el entorno del que hemos copiado el contenido. Para simplificar, diremos que al nuevo entorno se le ha modificado dicho campo y ahora es* MyMPI*. Una vez ya modificado ese campo con un editor de texto cualquiera, se añade este nuevo entorno paralelo a los gestionados por SGE con este comando:*

```
$ qconf -Ap  /nuestra/ruta/ejemplo/a/MyMPI.txt
```

El gestor de tareas debería avisar de que el entorno paralelo MyMPI *ha sido añadido correctamente, por lo que ahora se puede modificar el resto de parámetros del entorno paralelo con este comando:*

```
$ qconf -mp  MyMPI
```

Para el correcto funcionamiento del benchmark *HPL hay que cambiar en el entorno paralelo MPI corresponidente los parametros* control_slaves *a* TRUE *y* job_is_first_task *a* FALSE*. Ahora que ya se tiene un entorno paralelo, se puede definir un grupo de equipos. Por defecto y de forma transparente al usuario se creará siempre un grupo llamado* @allhosts*, lo que se puede ver con el siguiente comando que da un listado de todos los grupos que maneja SGE:*

```
$ qconf -shgrpl
```

Y si se quiere ver en detalle las características de un grupo de equipos (realmente un grupo de equipos se define con un nombre y un listado de nodos), se puede utilzar un comando parecido al anterior. Por ejemplo, aplicado a @allhosts*:*

```
$ qconf -shgrp @allhosts
```

De forma similar a los entornos paralelos, creamos nuestro propio grupo de equipos con los que trabajaremos de la siguiente forma:

```
$ qconf -shgrp @allhosts > /nuestra/ruta/ejemplo/a/MyHostGroup.txt
```

Se edita el campo group_name *(por ejemplo, llamamos a este nuevo grupo* @MyHostGroup*), y lo añadimos a los gestionados por SGE y modificamos (si se desea) con los siguientes comandos:*

```
$ qconf -Ahgrp /nuestra/ruta/ejemplo/a/MyHostGroup.txt
$ qconf -mhgrp @MyHostGroup
```

Ahora solo queda crear y configurar una cola de trabajo alternativa a la all.q *que se nos proporciona por defecto. Dicha cola contiene siempre a todos los nodos del clúster, por lo que puede interesar crear colas con un subconjunto de los nodos que compartan ciertas características, o bien modificar la carga capaces de soportar (por defecto están en 1.75, pero es recomendable reducir dicha carga máxima permitida con el fin de evitar que se sobrecalienten/saturen/caigan los nodos), o bien modificar cualquiera de las muchas opciones de la cola de trabajo. A continuación se muestran tres comandos: el primero sirve para ver todas las colas de trabajo gestionadas por SGE, el segundo es para mostrar en detalle la configuración de una cola en concreto, y el tercero para copiar la configuración de una cola en un archivo. Se tomará como ejemplo la cola* all.q*:*

```
$ qconf -sql
$ qconf -sq all.q
$ qconf -sq all.q > /nuestra/ruta/ejemplo/a/MyCola.txt
```

Se deberá modificar con un editor de textos el campo qname *por otro nombre que no sea* all.q*, por ejemplo* MyCola.q*. Despues se añade a las colas gestionadas por SGE y se modifican el resto de campos que se deseen con los dos siguientes comandos:*

```
$ qconf -Aq /nuestra/ruta/ejemplo/a/MyCola.txt
$ qconf -mq MyCola.q
```

Es muy importante que los campos hostlist *y* slots *coincidan con el grupo de equipos correspondiente y los nodos que dicho grupo gestione. Si se ha creado un nuevo entorno paralelo, se deberá asegurar que está en el campo* pe_list*.*
Ya se tiene entonces un grupo de nodos, un entorno paralelo, y una cola de trabajo con la que trabajar. Sin entrar en detalles, para ejecutar un programa paralelo se deben tener en cuenta dos aspectos. El primero es que SIEMPRE se ejecute con una cuenta que no sea la cuenta de root*. El segundo aspecto es que siempre se debería ejecutar el programa de forma no directa (ya que se intentaría ejecutar en el propio* frontend *y no en paralelo en los nodos), sino por medio de un script. Por ejemplo, tenemos un programa paralelo llamado* MyProg*, y los elementos gestionados por SGE antes descritos (entorno paralelo, grupo de nodos, y cola de trabajo). A su vez, suponemos que el grupo de nodos con el que trabaja está compuesto por 2 equipos los cuales son capaces de manejar 4 hilos cada uno. El script mínimo (sin entrar en opciones extra) que posibilita el que se pueda ejecutar en paralelo podría ser como esto:*

```
#!/bin/bash
# Opciones Predeterminadas
# Esto indica que se quiere trabajar en el directorio actual
#$ -cwd
```

```
# Esto indica la shell a utilizar
#$ -S /bin/bash
# Cola de trabajo a utilizar
#$ -q MyCola.q
# Entorno paralelo a utilizar con el numero de hilos a coordinar
#$ -pe MyMPI 8
#
# Ejecucion paralela del programa. np indica el numero de hilos necesario
mpirun  -np 8 MyProg
```

El 8 que aparece en dicho script es el resultado de multiplicar el número de equipos disponible por el número de hilos que es capaz de manejar cada uno. Suponiendo que hemos llamado a dicho script MyScript.sh*, se consigue hacer funcionar dicho script para la ejecución de un programa en el cluster de la siguiente forma:*

```
$ qsub /nuestra/ruta/ejemplo/a/MyScript.sh
```

Con el comando qstat *(y sus correspondientes opciones) se puede ver el estado del clúster, y de los trabajos en ejecución o pendientes de ejecución que se han enviado al clúster. En caso de necesitar revocar la ejecución de un trabajo en concreto se puede utilizar el comando* qdel *acompañado por el identificador de la tarea.*

Clústers Virtuales: *En esta subsección se tratará todo lo relativo a la creación y manejo de un clúster virtual por medio de Rocks Cluster. Para ello se harán dos diferencias de* frontend*: el nativo y el/los virtual/es. Estamos en un punto en el que se supone que tenemos un* frontend *nativo, el paquete KVM instalado (necesario instalarlo al instalar el* frontend *nativo) y una serie de contenedores (con el nombre asignado* vm-container-X-Y*). Como todo lo relativo a lo que es la configuración y gestión del clúster, casi todas las tareas se realizan por medio del comando* rocks*. Primero es necesario "preparar" la configuración del clúster virtual. Sirva de ejemplo el siguiente comando:*

```
$ rocks add cluster container-hosts="vm-container-0-0 vm-container-0-1" \
  cpus-per-compute=2  fe-name=frontend-0-0-0 ip=[IP Frontend Virtual] \
  mem-per-compute=2048 num-computes=8
```

Con el ejemplo anterior se "avisa" de que se quiere crear un cluster virtual que hace uso de dos contenedores virtuales (vm-container-0-0 vm-container-0-1) para crear 8 máquinas virtuales con 2GB de RAM y 2 unidades de procesamiento cada una. Este clúster virtual se identificará con el nombre de frontend-0-0-0 *y con una determinada IP asignada (no mostrada aquí, pero debe ser una dirección disponible). Ahora se debe proceder a crear unas claves que permitan al cluster físico y virtual comunicarse entre sí y acceder a comandos e información sensible alojadas en el otro clúster (aunque esto no siempre funciona). En el ejemplo siguiente se crea por medio de una sola línea tanto la clave privada como la pública sin necesidad de introducir una contraseña (que de otro modo nos estaría solicitando a cada momento):*

```
$ rocks create keys key=private00.key passphrase=no | tail -n 6 > public00.key
```

Como se ve a la clave privada se le ha llamado private00.key *y a la publica* public00.key*. No obstante el clúster virtual no ha sido realmente creado, solo ha sido preparado el entorno para su instalación. Para realmente crear el clúster virtual se necesita añadir la clave publica a esta preparación del cluster, encender el* frontend *virtual, y acceder al mismo. Esto se puede hacer ejecutando desde el* frontend *nativo con la cuenta de* root *los siguientes comandos:*

```
$ rocks add host key frontend-0-0-0 key=public00.key
$ rocks set host power frontend-0-0-0 action=install key=private00.key
$ rocks open host console frontend-0-0-0 key=private00.key
```

Para ejecutar este último comando se recuerda que si se está ejecutando en remoto, habrá que añadir al comando ssh *la opción* -X*, ya que este último comando abre una pantalla de acceso remoto al nuevo clúster que permite realizar la instalación gráfica del* frontend *virtual. Esta instalación es similar a la del* frontend *nativo, exceptuando que hay que realizar una instalación en red e introducir la IP (o el identificador único, el nombre del clúster en Internet) como origen del software a instalar, y que la red privada no debe ser igual que la red privada creada para la comunicación del clúster nativo. Una vez finalizada la instalación hay que volver a arrancar el* frontend *virtual ya que al finalizar su instalación, este se apaga.*

```
$ rocks set host power frontend-0-0-0 action=on key=private00.key
```

Una vez encendido se puede acceder al mismo tanto por ssh *como por medio del comando* rocks open host console frontend-0-0-0 key=private00.key*. Ya una vez en el frontend virtual se debe configurar el acceso a internet y un nuevo usuario tal y como se hizo en el clúster nativo. Para configurar los nodos virtuales, desde el* frontend *virtual se debe ejecutar* insert-ethers *al igua que se hizo en el clúster nativo y desde el* frontend *nativo arrancar las máquinas virtuales que serán los nodos del clúster virtual. Esto se hace consultando las MAC asignadas a las máquinas virtuales, y arrancando dichas direcciones MAC con los comandos siguientes (el primero es para consultar las direcciones, y el segundo para arrancarlo):*

```
$ rocks list macs host frontend-0-0-0 key=private00.key
$ rocks set host power [Direccion MAC] action=on key=private00.key
```

Ahora ya se tiene un clúster virtual, cuyo manejo es exactamente el mismo que un clúster nativo.

Eliminación de nodos y clústers virtuales: *La experiencia nos dicta que no es bueno realizar trasiegos de nodos, ni reconfiguraciones, y en el caso de los clústers virtuales, apagados y encendidos. Es mejor seguir un esquema en el que se cree/configure el clúster virtual, se use y una vez finalizado el uso, eliminar el clúster virtual. Por eso esta subsección está enfocada al correcto apagado y eliminación tanto de máquinas como de clústers virtuales.*
En cualquier caso, primero se apaga correctamente la/s máquina/s (ya sean físicas o virtuales), y luego se eliminan del esquema lógico del clúster. Para realizar el apagado se puede utilizar ssh [NOMBRE MAQUINA] 'shutdown -h now' *desde el* frontend *correspondiente., y para eliminar la máquina, el siguiente comando:*

```
$ rocks remove host [NOMBRE MAQUINA]
```

En el caso de que se quiera eliminar el clúster virtual, después de eliminar los nodos virtuales desde el frontend *virtual, se ejecuta:*

```
$ rocks remove cluster frontend-0-0-0
```

En cualquier caso despues de eliminar la/s máquina/s o el clúster correspondiente se debe ejecutar el siguiente comando para reconfigurar el clúster nativo:

```
$ rocks sync config
```

E Datos relativos a la prueba R1

Este apéndice contiene las gráficas resultantes en cuanto a rendimiento y tiempos obtenidos en la prueba R1. Los rendimientos estan medidos en GFlops y los tiempos en segundos. En el eje de abcisas se muestra la variación del tamaño del bloque (Block Size NB) en el algoritmo de resolución del HPL. Hay una gráfica de tiempos y rendimiento para cada una de las rejillas PxQ probadas (1x32, 2x16 y 4x8) y se muestra en cada una de ellas una curva para cada una de las combinaciones [HT desactivado — HT activado] y problema de [10k, 20k, 40k, 50k, 60k] índices de lado.

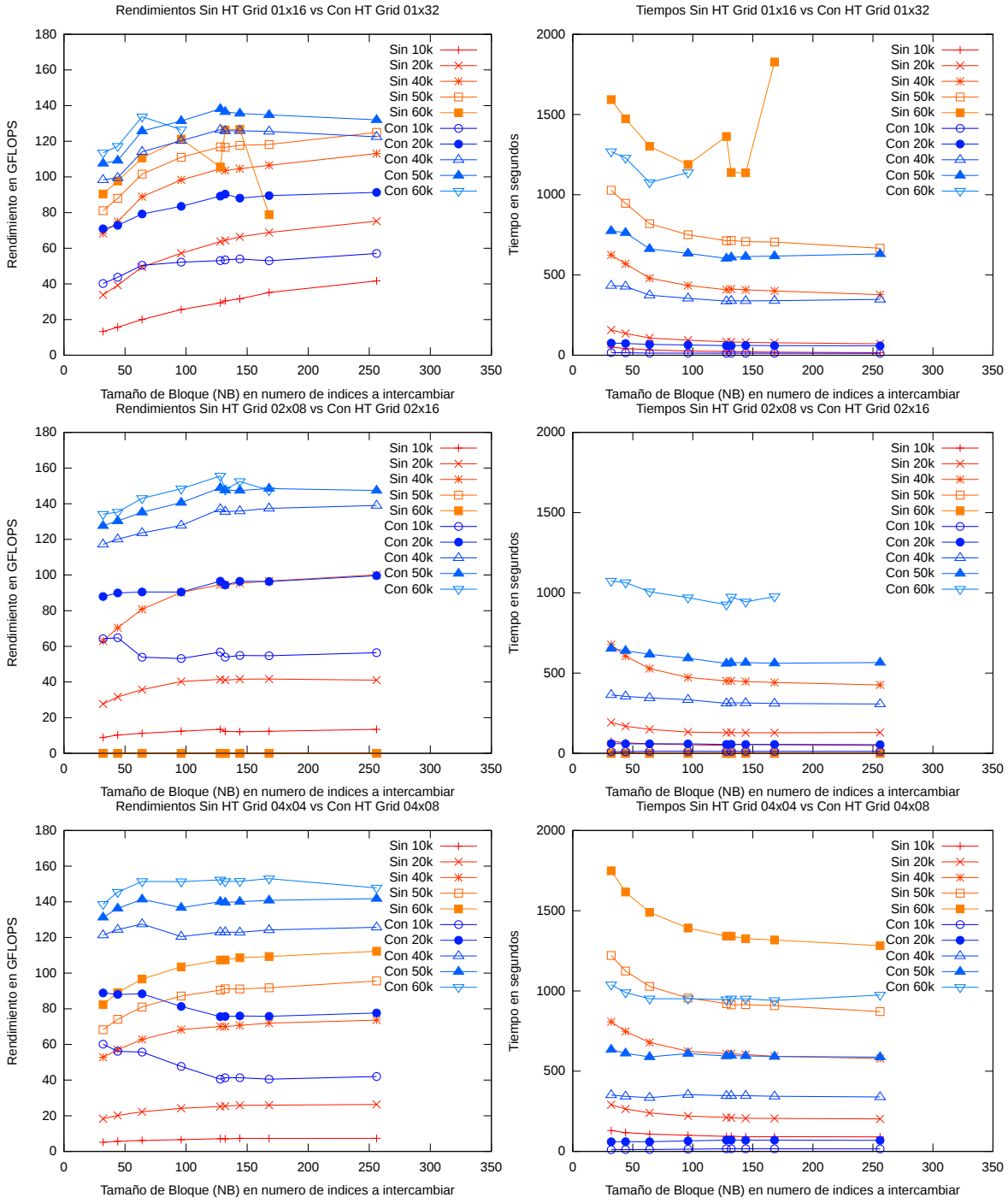

Fig. 13. Rendimientos y Tiempos de la prueba R1. Arriba utilizando una rejilla PxQ de 1x32, en medio una rejilla de 2x16, y abajo una de 4x8

F Datos relativos a la prueba V1

Este apéndice contiene las gráficas resultantes en cuanto a rendimiento y tiempos obtenidos en la prueba V1. Los rendimientos estan medidos en GFlops y los tiempos en segundos. En el eje de abcisas se muestra la variación del tamaño del bloque (Block Size NB) en el algoritmo de resolución del HPL. Hay una gráfica de tiempos y rendimiento para cada una de las rejillas PxQ probadas (1x32, 2x16 y 4x8).

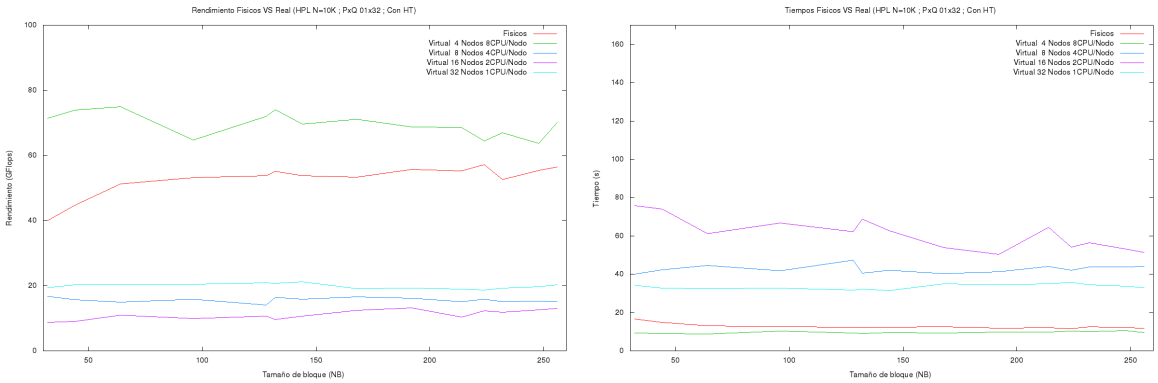

Fig. 14. Rendimientos y Tiempos utilizando una rejilla PxQ de 1x32.

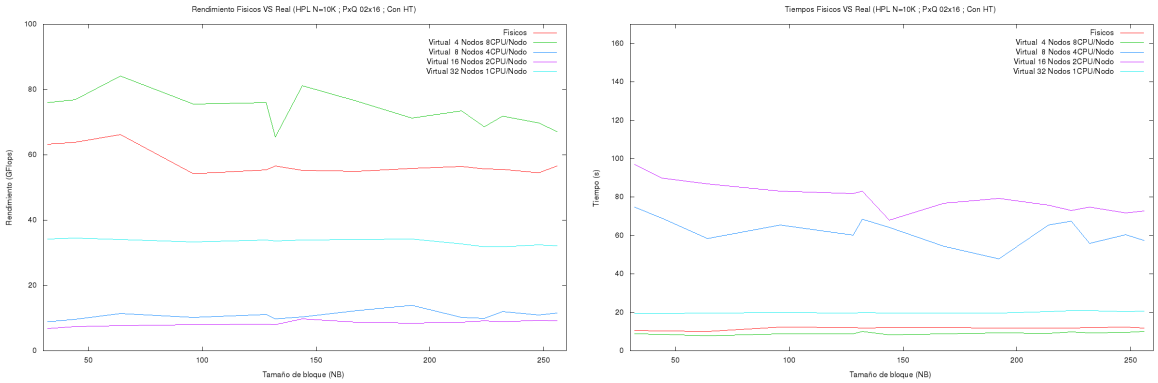

Fig. 15. Rendimientos y Tiempos utilizando una rejilla PxQ de 2x16.

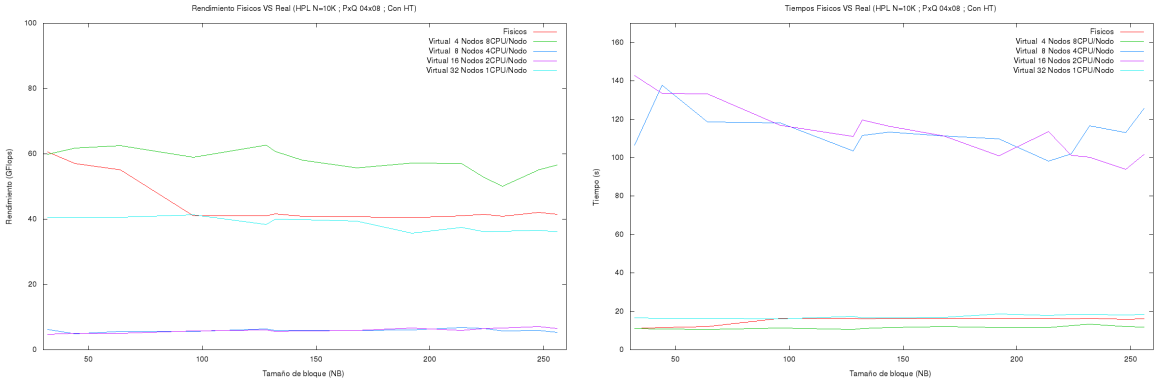

Fig. 16. Rendimientos y Tiempos utilizando una rejilla PxQ de 4x8.

G Datos relativos a la prueba V2

Este apéndice contiene los datos resultantes obtenidos en la prueba V2. Se muestran las gráficas para las distintas configuraciones del cluster[57]*, primero se muestran para cada una de las configuraciones, los resultados obtenidos para cada una de las rejilla de procesos PxQ*[58] *a lo largo de las distinta configuraciones de N y NB de HPL. Los rendimientos estan medidos en GFlops en el eje de ordenadas y en el eje de abcisas se muestra la variación del tamaño del bloque (Block Size NB) en el algoritmo de resolución del HPL. Hay una gráfica de tiempos y rendimiento para cada una de las rejillas PxQ probadas .*

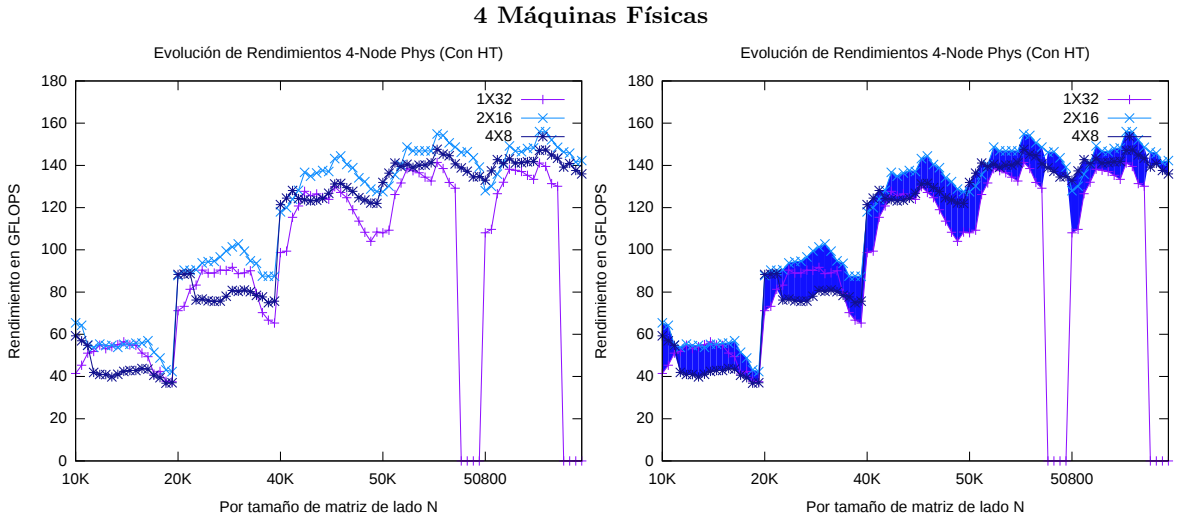

Fig. 17. Rendimientos obtenidos por el cluster computesto por 4 Máquinas Físicas para las distintas rejillas PxQ.

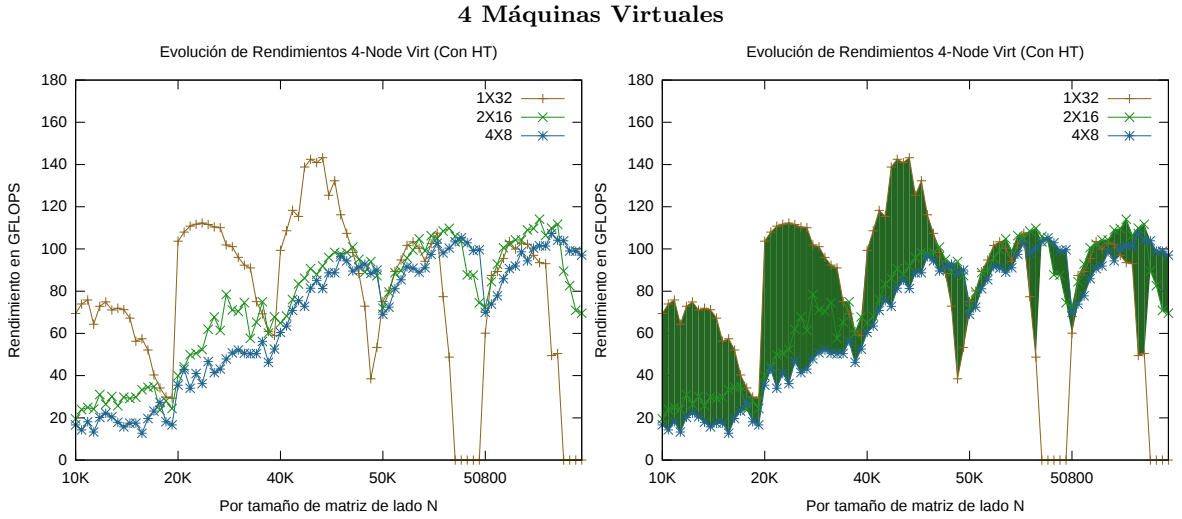

Fig. 18. Rendimientos obtenidos por el cluster computesto por 4 Máquinas Virtuales para las distintas rejillas PxQ.

[57] 4 máquinas físicas, y 4, 8 16 y 32 máquinas virtuales utilizando esas mismas 4 máquinas físicas.

[58] Las pruebas se centraron en (1x32, 2x16 y 4x8) para todas las configuraciones de cluster virtual, no obstante se muestra para algunas pruebas para algunas pruebas

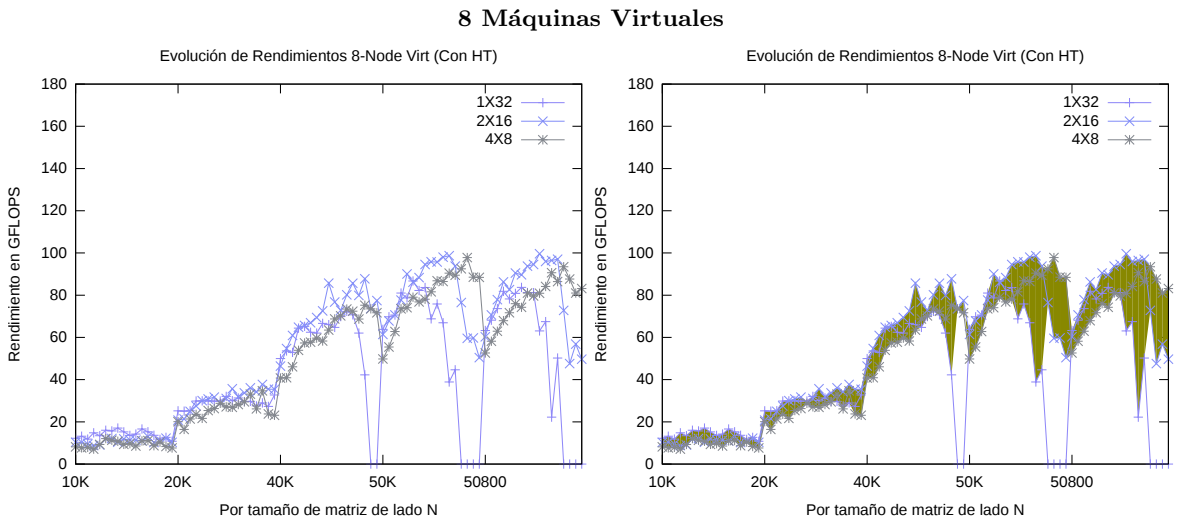

Fig. 19. Rendimientos e intervalos obtenidos por el cluster computesto por 8 Máquinas Virtuales para las distintas rejillas PxQ.

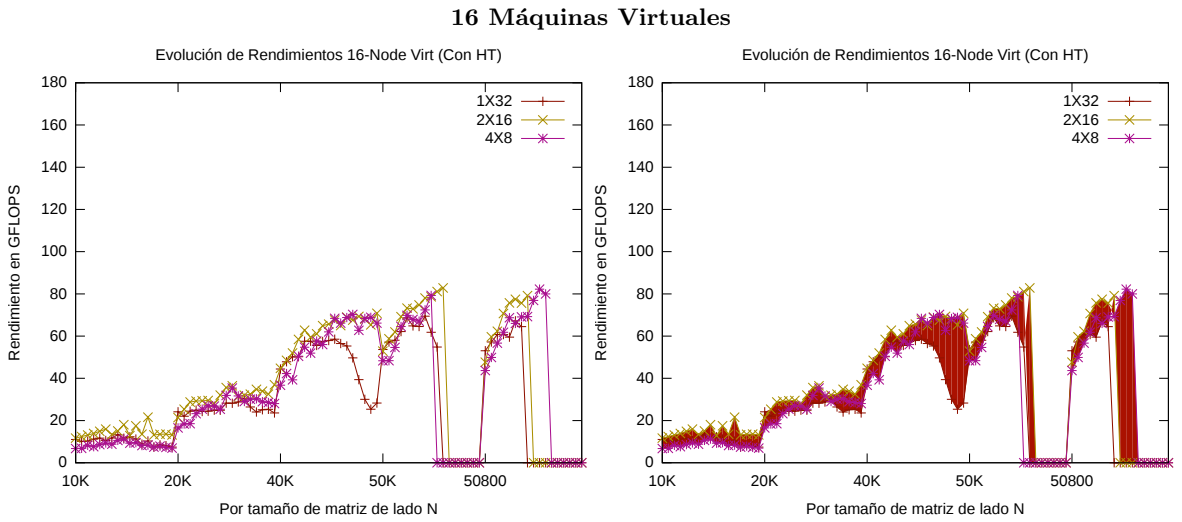

Fig. 20. Rendimientos e intervalos obtenidos por el cluster computesto por 16 Máquinas Físicas para las distintas rejillas PxQ.

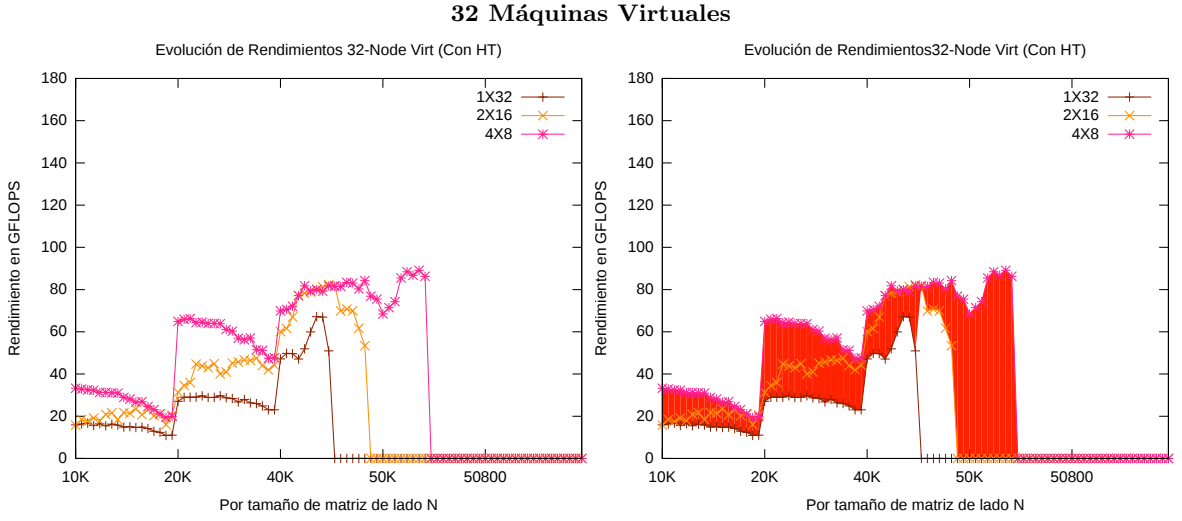

Fig. 21. Rendimientos e intervalos obtenidos por el cluster computesto por 32 Máquinas Físicas para las distintas rejillas PxQ.

Selección de los mejores resultados según configuración y tamaño de problema (de lado N, en indices de la matriz a resolver)

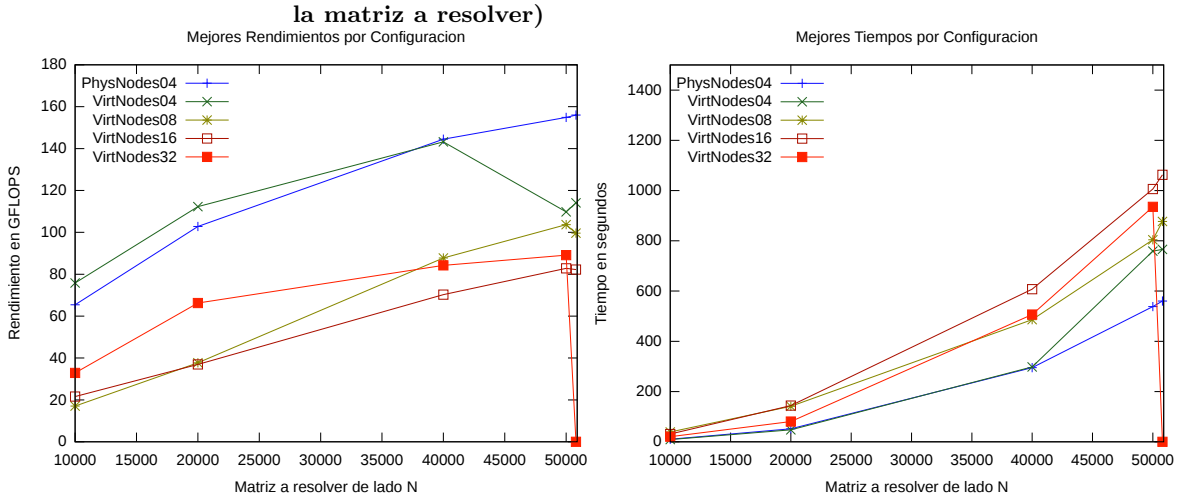

Fig. 22. Mejores rendimientos y tiempos obtenidos por cada configuración de cluster

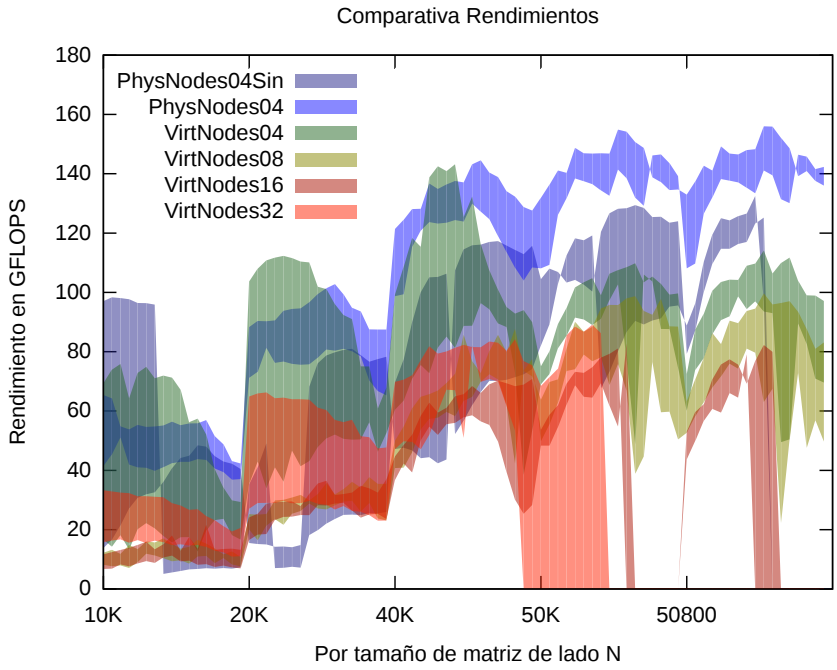

Fig. 23. Intervalos de rendimientos (según las distintas rejas PxQ utilizadas) para los distintos tamaños de problema probados

Selección de los mejores rendimientos (y tiempos asociados) por tamaño de problema y reja de procesos PxQ utilizada

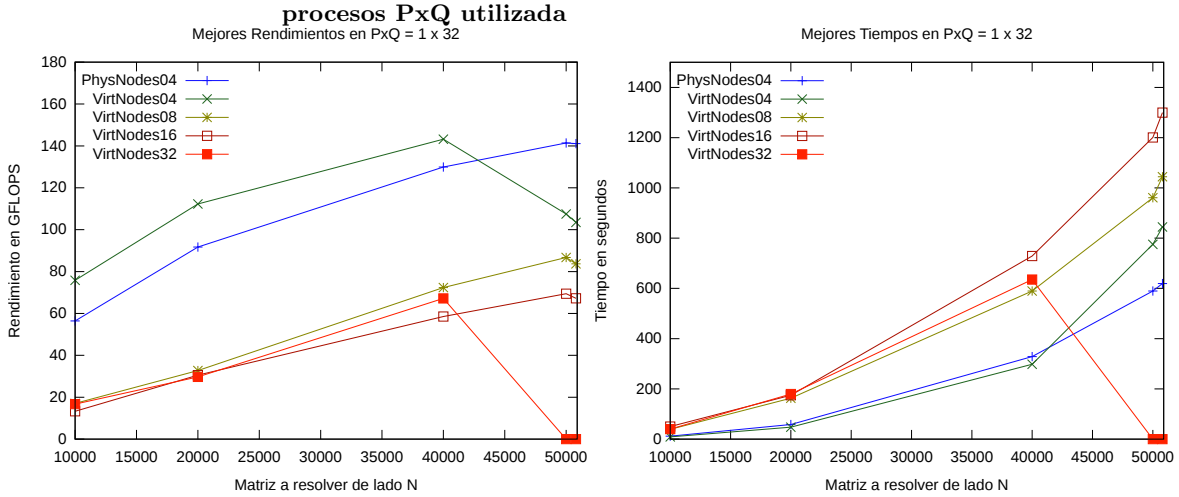

Fig. 24. Rendimientos y Tiempos utilizando una rejilla PxQ de 1x32.

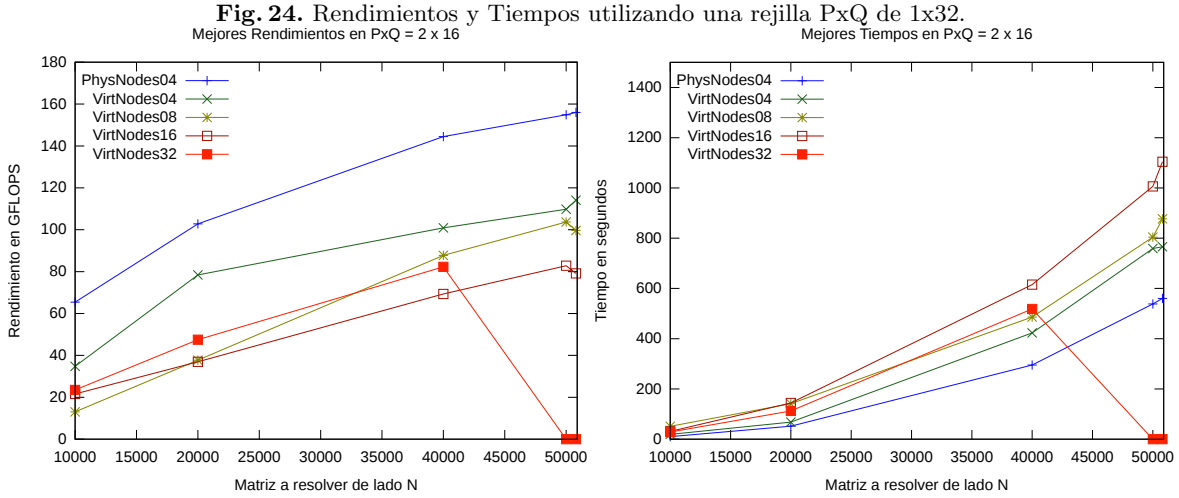

Fig. 25. Rendimientos y Tiempos utilizando una rejilla PxQ de 2x16.

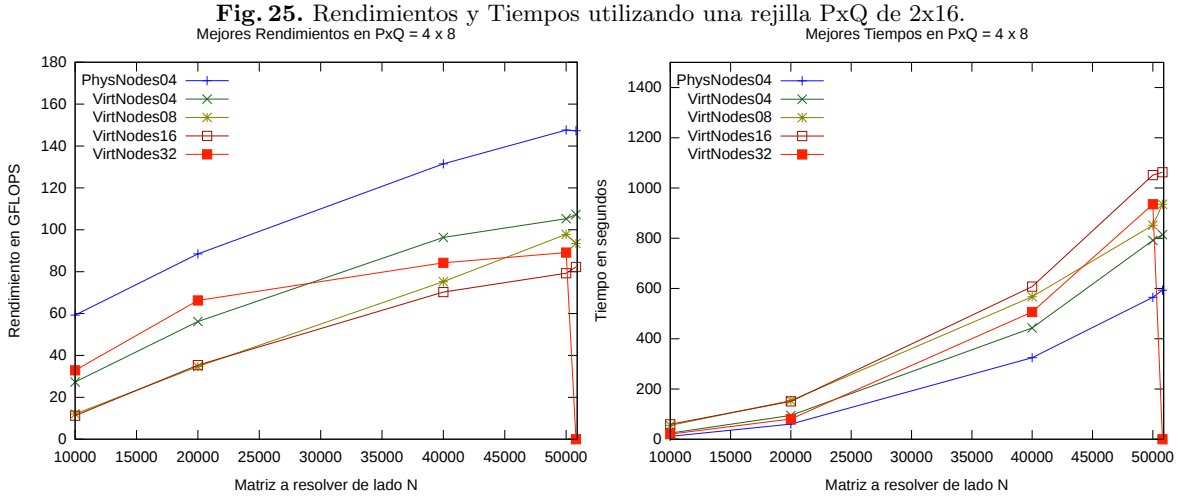

Fig. 26. Rendimientos y Tiempos utilizando una rejilla PxQ de 4x8.

Selección de los mejores resultados según configuración y porcentaje de memoria utilizado

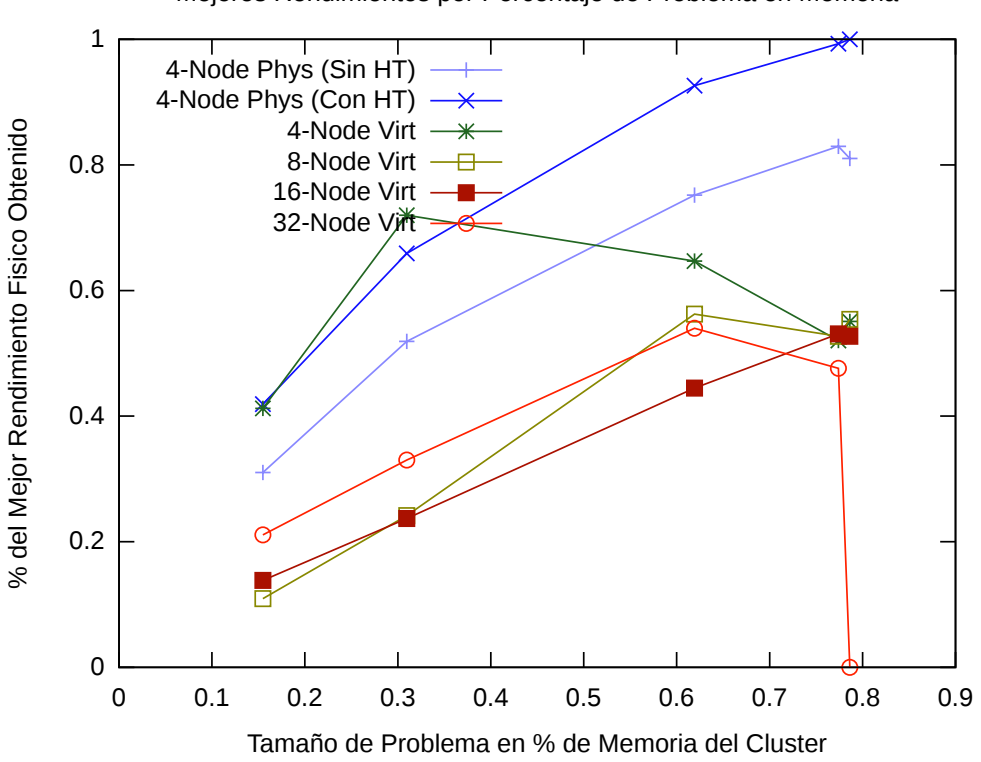

Fig. 27. Esta gráfica expresa los mejores resultados obtenidos por cada configuración del cluster probada (por cada uno de los tamaños de problema utilizados, expresados en porcentaje de memoria principal del cluster utilizada) en base al porcentaje del mejor rendimiento obtenido por el cluster físico (para cualquier tamaño de problema probado).

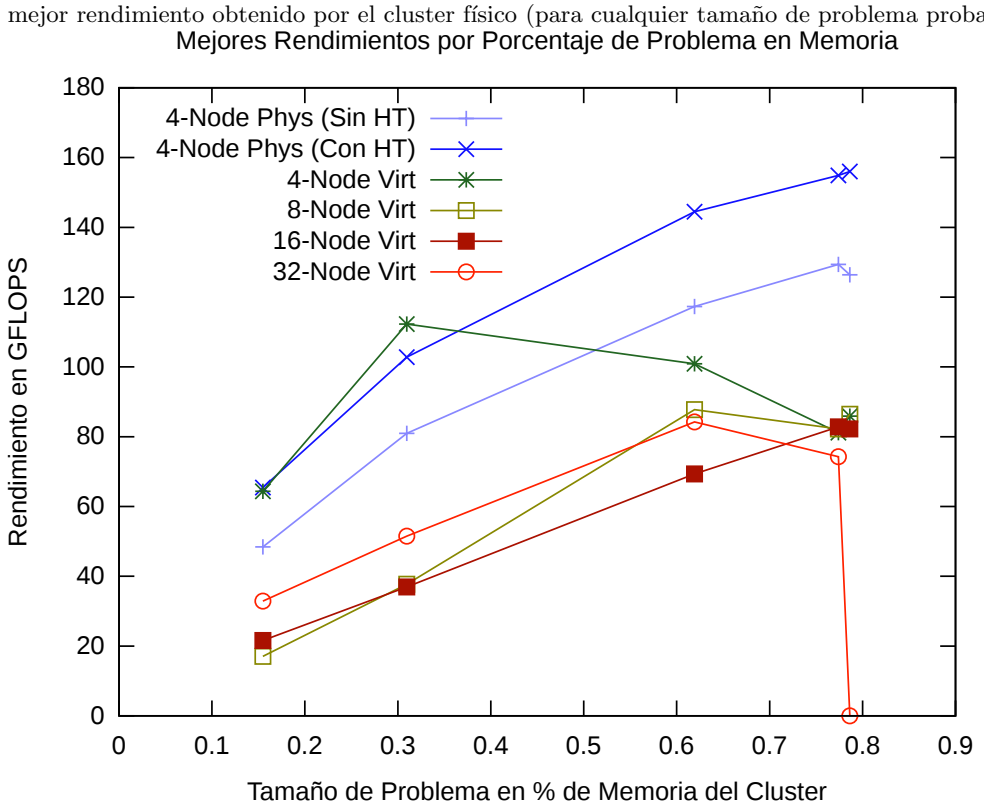

Fig. 28. Esta gráfica expresa los mejores resultados obtenidos por cada configuración del cluster probada (por cada uno de los tamaños de problema utilizados, expresados en porcentaje de memoria principal del cluster utilizada) en GFLOPS.

H Datos relativos a la prueba V3

Este apéndice contiene los datos resultantes obtenidos en la prueba V3. Se muestran las gráficas para las distintas configuraciones del cluster[59]*, utilizando las rejillas de procesos PxQ 1x8 y 2x4 (ya que se utilizan 8 cores lógicos para esta prueba). Primero se muestran los resultados para cada una de las configuraciones del cluster, seguidamente se muestran los resultados por rejilla PxQ utilizada y por ultimo se muestra la gráfica que aúna las variaciones de configuración de cluster y las rejas PxQ utilizadas (ya mostrado en el propio desarrollo de la prueba V3). Para todos los problemas se utilizan N igual a 10000 y 20000 (recordemos que N es el lado en índices de la matriz cuadrada a resolver), y NB que van desde 32 a 1024 (tamaño de mensaje a utilzar entre los distintos proecesos que componen la prueba, en índices tambien).*

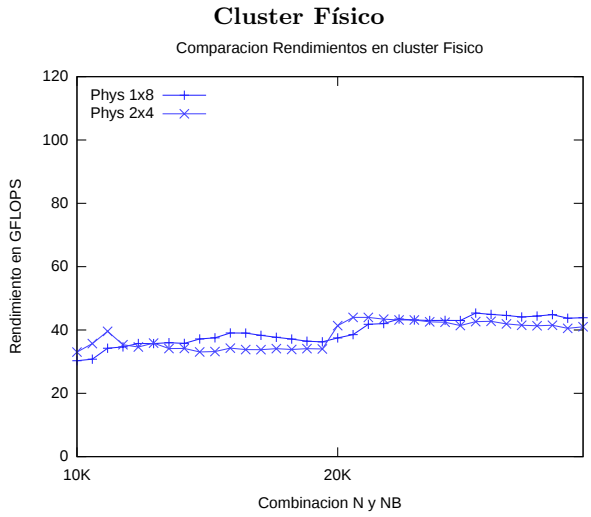

Fig. 29. Rendimientos obtenidos por el cluster físico computesto por 1 Máquina Física con 8 cores lógicos para las distintas rejillas PxQ.

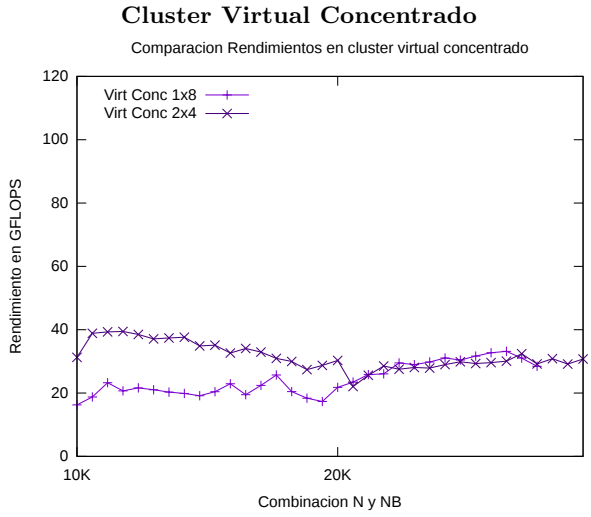

Fig. 30. Rendimientos obtenidos por el cluster virtual computesto por 8 Máquinas Virtuales (cada una de 1 core lógico) en 1 Máquina Física para las distintas rejillas PxQ.

[59] Un cluster físico en 1 máquina física usando 8 cores lógicos (al cual nos referimos como Cluster Físico), un cluster virtual en 8 máquinas virtuales en una misma máquina física (al cual nos referimos como Cluster Virtual Concentrado), y un cluster virtual en 8 máquinas virtuales en 8 máquinas físicas distintas (al cual nos referimos como Cluster Virtual Disperso).

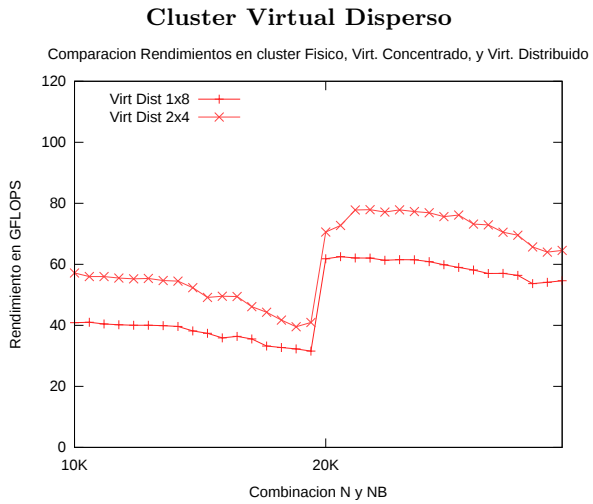

Fig. 31. Rendimientos obtenidos por el cluster virtual computesto por 8 Máquinas Virtuales (cada una de 1 core lógico) 8 Máquinas Físicas para las distintas rejillas PxQ.

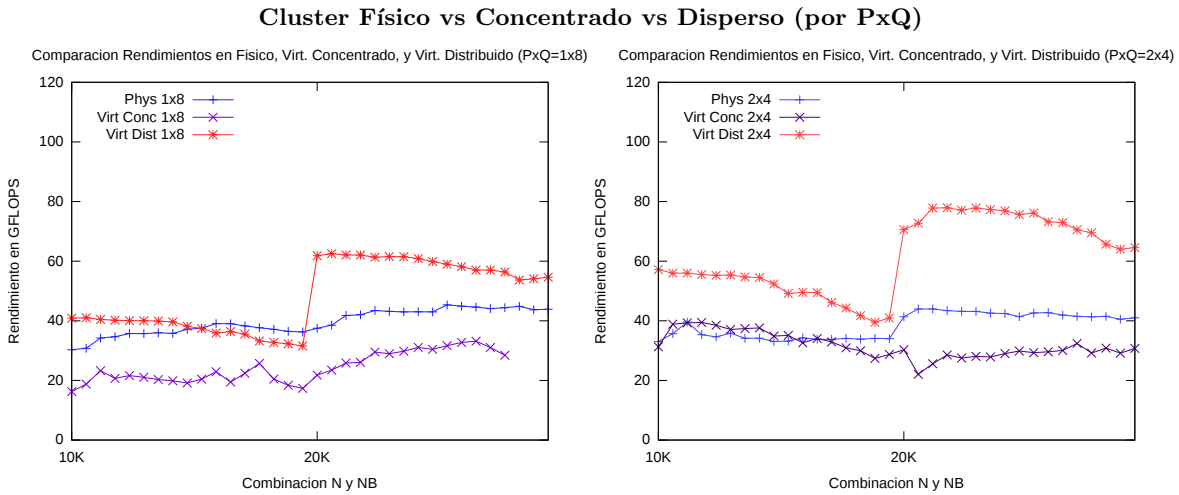

Fig. 32. Rendimientos obtenidos por los clusters probados para las rejillas 1x8 y 2x4.

I Glosario

En esta sección se explica términos de uso frecuente en este documento, aclarando aquellos que pueden llevar a confusión (por ser de uso común, coindidir con términos que se usen en otros campos, etc ...), cuales evitaremos y cuales intentaremos utilizar, tratando así de establecer un lenguaje coherente de cara a este trabajo (al margen de que pueda haber otras definiciones). A algunos términos les acompaña su abreviatura (entre corchetes).

Clúster: Conjunto de máquinas (**nodos**) que comparten sus recursos actúando como una única máquina, de forma que se aumentan los recursos del sistema en algún aspecto clave (como pueda ser la capacidad de cómputo en términos de velocidad o tamaño del problema a resolver). Normalmente uno de los nodos realiza las funciones de coordinación del resto de máquinas, de conexión de comunicaciones con el exterior, y como interfaz del clúster (véase **frontend**). El resto de nodos aporta sus recursos al clúster, ya sea para realizar las tareas de computación necesarias (**nodo de computación**) o como instrumento para almacenar y ejecutar **máquinas virtuales** (**contenedor de máquinas virtuales**). Véase **clúster físico** y **clúster virtual**.

Clúster Físico: **Clúster** compesto únicamente por **máquinas físicas**. También conocido como **clúster nativo**.

Clúster Nativo: Véase **Clúster Físico**.

Clúster Virtual: **Clúster** cuyos **nodos de computación** son **máquinas virtuales**. El **frontend** puede ser también una máquina virtual (lo más habitual).

Cola de Trabajo: Es una herramienta que permite al **clúster** ejecutar tareas de forma coordinada en varios nodos del mismo. La configuración de cada cola de trabajo incluye un determinado grupo de trabajo con sus **slots** asociados, los usuarios permitidos, el entorno paralelo (es decir, la forma en la que se paraleliza el problema y se sincronizan los procesos; en nuestro caso **MPI**), la carga limite por nodo permitida y otros parámetros. Al igual que permite ejecutar tareas de forma paralela, es el encargado de monitorizar y eliminar procesos (llegado el caso).

Compute Node: Véase **Nodo de Computación**.

Contenedor de Máquinas Virtuales, **Contenedor** o también **VM-Container**: **Nodo** del **clúster** compuesto por una **máquina física** que tiene como único prósito alojar y ejecutar **máquinas virtuales**. Las máquinas virtuales que sean alojadas y ejecutadas en un contenedor por regla general pertenecerán a un **clúster virtual** que será independiente del **clúster físico** al que pertenece el contenedor de dichas máquinas virtuales.

Core : Unidad de procesamiento. Debido a que puede haber diferencias entre las unidades de procesamiento físico (**cores físicos** que tiene un **microprocesador** y las unidades de procesamiento que ve el sistema operativo (**cores lógicos**) se evitará utilizar este término sin su respectiva aclaración, aunque debido a que utilizamos el sistema operativo para tratar con la infraestructura física, si este término se encuentra solo, es posible que estemos hablando de **cores lógicos**.

Core Físico [**PhyCore**]: El número de las unidades en un solo chip (**microprocesador**) capaces de ejecutar procesos. Es desde un punto de vista hardware (totalmente físico, las unidades

capaces de ejecutar procesos grabadas en el chip).

Core Lógico [**LogCore**]: Desde el punto de vista software (del sistema operativo), el número de las unidades capaces de ejecutar procesos en una máquina (independientemente de que ésta sea física o virtual). Lo habitual es que el sistema operativo vea el mismo número de cores físicos y los interpeten como tales, no obstante, hay técnicas que hacen que esto no siempre sea así, pudiendo el sistema operativo interpetar que hay más unidades de procesamiento independientes de las que hay (vease ***HyperThreading***).

CPU: Unidad de procesamiento. En este trabajo equivalente al término **Core** (debido a que ambos y por los mismos motivos, tendemos a no utilizar). Véase **Core Físico**, y **Core Lógico**.

Entorno Paralelo [**PE**]: Es forma predefinida que se usa para paralelizar un problema y ser capaz de coordinarlo entre varios procesos. En nuestro caso utilizamos **MPI**.

FE: Véase **Frontend**.

Frontend [**FE**]: **Nodo** del **clúster** que actua como interfaz del mismo, coordinador del resto de nodos (a través del **gestor de recursos**)y de interconexión con el exterior. Su naturaleza puede ser tanto virtual como física.

Gestor de Tareas: También conocido como **Gestor de Colas de Trabajo** Es el software encargado de gestionar las **colas de trabajo**, asociando **nodos de computación** y sus **slots**, a un determinado **entorno paralelo** y permitiendo su uso de forma coordinada. En nuestro caso usamos **SGE**.

Gestor de Colas de Trabajo: Vease **Gestor de Tareas**.

Gestor de Recursos: Es el software encargado de gestionar los recursos de los distintos nodos del clúster Vease **Gestor de Tareas**.

Grupo de Trabajo [**Hostgroup**]: Subconjunto de **nodos de computación** manejado por el **gestor de tareas**.

Infraestructura Física: Término para referirnos al conjunto de hardware (**máquinas físicas** e interconexión de las mismas) que conforman un **clúster**.

High Performance Linpack [**HPL**]: Software basado en librerias de algebra lineal que sirve como banco de pruebas a sistemas dedicados a la compuación de altas prestaciones. Consiste en la resolución de un sistema de ecuaciones mediante la factorización LU de la matriz (cuadrada) asociada.

HPL: Véase **High Performance Linpack**.

HT: Véase ***HyperThreading***.

HyperThreading [**HT**]: Es una tecnología de Intel que permite ofrecer dos hilos de procesamiento por cada **core físico** dentro de un **microprocesador**. De cara a este trabajo supone que hace ver al sistema operativo el doble de cores que realmente tiene el microprocesador (de ahí la diferencia entre **cores físico** y **cores lógico**).

Kernel Virtual Machine [**KVM**]: Conjunto de software para GNU/Linux que permite virtualizar una **máquina física**.

KVM: Véase **Kernel Virtual Machine** [**KVM**].

LogCore: Véase **core lógico**.

Message Passing Interface [**MPI**]: Es un sistema de comunicación entre procesos por medio de una pasarela de mensajes. Al paralelizar el programa, se divide el problema entre tantos procesos como **slots** (cores) se hayan solicitado y los mismos se comunican entre sí de para compartir información por medio de mensajes.

Microprocesador: Encapsulado físico, modelo comercial de chip. Un microprocesador puede contener varios **CPUs** o **cores** a nivel *físico*.

MPI: Véase **Message Passing Interface**.

Máquina Física [**PhM**]: Conjunto de hardware que es capaz de ejecutar procesos, comunicarse con otras máquinas y es gestionado por un sistema operativo.

Máquina Virtual [**VM**]: Conjunto de software que actúa como una **máquina física**. Dicho sofware realmente gestiona los recursos de la máquina física (a través de la tecnología de virtualización tanto hardware como software correspondiente).

NB: Véase **Tamaño de bloque**.

N: Véase **Tamaño del problema**.

Nodo: Máquina (ya sea física o virtual) que forma parte del **clúster**. Dependiendo de su función, ya sea como **Frontend** (servidor y nodo principal del clúster), **Contenedor de VM**, o **nodo de computación**. Véase **Frontend**, **Contenedor de Máquinas Virtuales** y **Nodo de Computación**.

Nodo de Computación: También conocido como **Compute Node**, se refiere a la máquina que aporta al **clúster** capacidad de procesamiento. Puede ser tanto una **máquina física** como una virtual. En caso de tratarse de una **máquina virtual**, ésta puede estar alojada o bien en la misma máquina física que el **frontend**, en una máquina especialmente preparada a nivel software para tener como único propósito alojar y ejecutar máquinas virtuales (**contenedor de áquinas virtuales**), o incluso alojarse en otras máquinas físicas accesibles por red no pertenecientes al clúster pero que permiten ejecutar máquinas virtuales.

Nodo Físico [**PhysNode**]: Nodo que se esta ejecutando directamente bajo una **máquina física**. Puede referirse tambien a un nodo virtual según su función dentro del clúster, por ejemplo **frontend físico** o **nodo de computación físico**, pero un **contenedor será siempre una máquinas física**.

Nodo Virtual [**VirtNode**]: Nodo que se esta ejecutando bajo una **máquina virtual**. Puede referirse tambien a un nodo virtual según su función dentro del clúster, por ejemplo **frontend virtual** o **nodo de computación virtual**, pero un **contenedor será siempre una**

máquinas física.

Parallel Environment [**PE**]: Véase **Entorno Paralelo**.

PhyCore: Véase **core físico**.

PhysNode: Véase **nodo físicol**.

Procesador Debido a la posible confusión que pueda haber entre **microprocesador** o unidad de procesamiento (tanto a nivel físico como lógico), o el termino , evitaremos utilizar este término en este trabajo.

Rejilla de procesos PxQ: Parámetro de **HPL** que indica la distribución de procesos que conforman la matríz a resolver. En ocasiones se le llama **rejilla PxQ**.

Rocks: No confundir con el comando *rocks* que es el comando principal de gestión de los clústers (tanto a nivel físico como virtual) en esta distribución. Véase **Rocks Cluster**.

Rocks Cluster [**Rocks**][60]: Distribución GNU/Linux basada en CentOS capaz de ejecutarse en varias máquinas tanto (físicas como virtuales) para conformar un clúster. Integra varias herramientas esenciales para el clustering (**MPI**, **SGE**, ...) y para la virtualización (**KVM**, **VMM**, ...).

Simultaneous multithreading: Tecnología a nivel hardware que permite el uso de los recursos provistos por un procesador a multiples hilos a la vez.

SGE: Váse **Sun GridEngine**.

Slot: Un slot es, desde el punto de vista de un **gestor de tareas** un**core lógico** asociado a un **nodo**.

SMT: Véase **Simultaneous multithreading**.

Sun GridEngine: Software que nos proporciona el servicio de **gestor de colas de trabajo**.

Tamaño de bloque [**NB**]: Parámetro de **HPL** que indica el tamaño de los mensajes a utilizar para la comunicacion entre procesos de la **rejilla de procesos PxQ** (expresado en número de elementos de la matriz a resolver). Los elementos de la matriz son de tipo coma flotante con doble precision (64 bits, 8 Bytes).

Tamaño del problema [**N**]: Parámetro de **HPL** que indica el tamaño de lado de la matriz cuadrada a resolver (expresado en número de elementos). Los elementos de la matriz son de tipo coma flotante con doble precision (64 bits, 8 Bytes).

Unidad de procesamiento: Véase **Core Lógico**.

VirtNode: Véase **nodo virtual**.

[60] NOTA: No confundir con el comando *rocks* que es el comando principal de gestión de los clústers (tanto a nivel físico como virtual) en esta distribución.

Virtual Machine Manager [**VMM**]: Interfaz gráfico para gestionar **máquinas virtuales**, en nuestro caso, bajo **KVM**.

VM-Container: Véase **Contenedor de Máquinas Virtuales**.

(Página en blanco)