

Índice de contenido

1. [Introducción](#)

- 1.1. [Motivación](#)
- 1.2. [Objetivos](#)
- 1.3. [Organización de la memoria](#)
- 1.4. [Planificación](#)

2. [Message passing interface \(MPI\)](#)

- 2.1. [Introducción](#)
- 2.2. [Comunicación punto a punto](#)
 - a) [Operaciones de envío bloqueantes](#)
 - b) [Operaciones de recepción bloqueantes](#)
 - c) [Comparación y conversión de tipos de datos](#)
 - d) [Comunicaciones no bloqueantes](#)
- 2.3. [Comunicaciones colectivas](#)
 - a) [Broadcast](#)
 - b) [Gather](#)
 - c) [Scatter](#)
 - d) [Gather to all](#)
 - e) [All to all](#)
- 2.4. [Los objetos communicator](#)

3. [Herramientas](#)

- 3.1. [KVM](#)
- 3.2. [Benchmarking](#)
 - a) [sysbench](#)
 - b) [b_eff_io](#)
 - c) [httperf](#)

3.3. [Hardware](#)

- a) [HDD](#)
- b) [SSD](#)
- c) [Fusion-io ioDrive duo 640Gb](#)
- d) [Potenciómetro](#)

4. [Preparación del entorno y ejecución de las pruebas](#)

- 4.1. [Instalación de KVM](#)
- 4.2. [Preparación de las máquinas virtuales](#)
- 4.3. [Instalación de iodrive](#)
- 4.4. [Scripts](#)

- a) [Script para las máquinas virtuales](#)
- b) [Script para benchmarking de HTTP](#)
- c) [Script potenciómetro](#)

4.5. [Instalando las herramientas de Benchmarking](#)

- a) [sysbench](#)
- b) [b_eff_io](#)
- c) [httperf](#)

4.6. [Ejecución de las pruebas](#)

- a) [httperf](#)
- b) [sysbench](#)
- c) [b_eff_io](#)

4.7. [Otras consideraciones](#)

5. [Resultados y conclusiones](#)

- 5.1. [httperf](#)
- 5.2. [sysbench](#)
- 5.3. [b_eff_io](#)
- 5.4. [Discusión](#)
- 5.5. [Trabajo futuro](#)

6. [Anexos](#)

6.1. [Anexo 1 - Resultados](#)

a) [httperf](#)

- [HDD + swap HDD](#)
- [HDD + swap SSD](#)
- [SSD + swap SSD](#)
- [Iodrive + swap SSD](#)

b) [sysbench](#)

- [HDD + swap HDD](#)
- [HDD + swap SSD](#)
- [SSD + swap SSD](#)
- [Iodrive + swap SSD](#)

c) [b_eff_io](#)

- [HDD + swap HDD](#)
- [HDD + swap SSD](#)
- [SSD + swap SSD](#)
- [Iodrive + swap SSD](#)

7. [Referencias](#)

1. Introducción

1.1. **Motivación**

En el mundo actual cada vez nos encontramos mas rodeados por la tecnología, y los servicios de la industria de las TIC son cada vez mas demandados. Por otro lado la sociedad es cada vez mas consciente del impacto que la actividad humana tiene sobre el medio ambiente. Las industrias de las TIC forman parte también de ese impacto, según un informe elaborado por la consultora Gartner las TIC suponen un 2% de las emisiones mundiales de dióxido de carbono.

Pero la preocupación por el consumo energético no viene solo motivada por la conciencia medio ambiental. El coste de la energía, cada día mas alto, hace que los clientes demanden cada vez mas mecanismos que sirvan para controlar este consumo energético, y las empresas desarrollan agendas de cambio que minimizan este tipo de costes.

A raíz de los motivos expuestos ha surgido el concepto de “Green IT” o informática verde, que trata de convertir la industria de las TIC en una industria sostenible. Uno de los pilares principales del Green IT es la virtualización, ya que permite contar con un menor número de máquinas físicas manteniendo la misma capacidad de procesamiento y aprovechando al máximo recursos de hardware, que hasta ahora se encontraban muy por debajo de su posible uso.

Cuando estas máquinas, ya sean virtuales o físicas, funcionan como sistemas distribuidos la demanda de potencia es alta . Para funcionar en paralelo estos sistemas disponen de una interfaz que permite mandar mensajes entre los diferentes nodos que la conforman, una instancia para computación de altas prestaciones es el modelo de programación MPI que permite todas las funcionalidades necesarias y esta ampliamente aceptado.

1.2. **Objetivos**

La idea del presente trabajo es: medir en estos sistemas distribuidos cuanto supone el coste energético con la utilización de máquinas virtuales en función de las tecnologías de almacenamiento empleadas, y determinar si existen posibles estrategias que mejoren este ahorro de consumo.

Este trabajo también persigue entender el comportamiento de diferentes tipos de aplicaciones en entornos virtualizados para modelar su comportamiento en cuando a tiempo de ejecución y coste energético así como entender el compromiso entre rendimiento y energía. A partir de este estudio se podrán definir modelos que podrán ser

usados para diseñar nuevas jerarquías de memoria con tecnologías emergentes como las basadas en memorias no volátiles.

Para ello se dispondrán diferentes recursos, tanto de software como de hardware, con los que procederemos a medir los consumos producidos en la ejecución de aplicaciones que precisen de una alta capacidad computacional. En el caso del presente trabajo nos centraremos en los sistemas distribuidos que empleen aplicaciones tanto MPI, como de bases de datos y servicios web.

La tecnología de virtualización o hipervisor elegido para realizar el estudio ha sido KVM, y a través de esta generamos el entorno virtualizado sobre el que trabajar. Esta tecnología de virtualización nos permitió emular un sistema distribuido y los diferentes tipos de servidores sobre los que realizaremos nuestras mediciones.

La comparativa entre estos sistemas distribuidos virtuales se realizó en base a la tecnología de almacenamiento sobre la que trabajaban, de manera que obtuviéramos una orientación sobre si estas tecnologías acercan a las TIC hacia ese ideal de industria sostenible y verde, que comentábamos en párrafos anteriores.

1.3. **Organización de la memoria**

La memoria se compone de 5 apartados y un anexo. El primer apartado, del que forma parte este subapartado, pretende exponer la motivación y organización del presente proyecto.

Los apartados 2 y 3 exponen todos aquellos elementos que se han empleado para este proyecto y pretende dar una descripción detallada de estas tecnologías y herramientas.

En el apartado 4 se recogen los pasos que se han seguido para tener un entorno de pruebas preparado junto con una exposición de como se han ejecutado todas las pruebas planteadas.

Finalmente en el apartado 5 se encuentran las conclusiones extraídas del conjunto del proyecto en base a los datos obtenidos de las pruebas. Estos datos pueden encontrarse en el anexo 1 que recoge el resultado de la ejecución de las pruebas tal y como lo proporcionaron las herramientas usadas.

1.4. **Planificación**

El proyecto se compuso de varias etapas:

Planificación: Inicialmente se desarrolló un plan de trabajo, que permitiera la consecución de resultados de la manera mas óptima barajándose todas las posibles opciones y procedimientos disponibles. Posteriormente los diferentes obstáculos encontrados obligaron a replantear esta planificación.

Preparación del entorno: Durante esta etapa se procedió a instalar y configurar tanto KVM como las diferentes máquinas virtuales que empleamos. También instalamos y configuramos las herramientas de medición¹ con las que evaluamos cada uno de los entornos preparados.

Toma de datos: Una vez que el entorno se encontró configurado completamente, procedimos a realizar las primeras pruebas con las herramientas seleccionadas. Para determinar la complejidad de las mediciones a realizar probamos diferentes combinaciones hasta dar con los parámetros que se ajustarán a los experimentos deseados. Una vez definidos claramente los experimentos procedimos a ejecutarlos, midiendo a su vez los consumos producidos durante su ejecución.

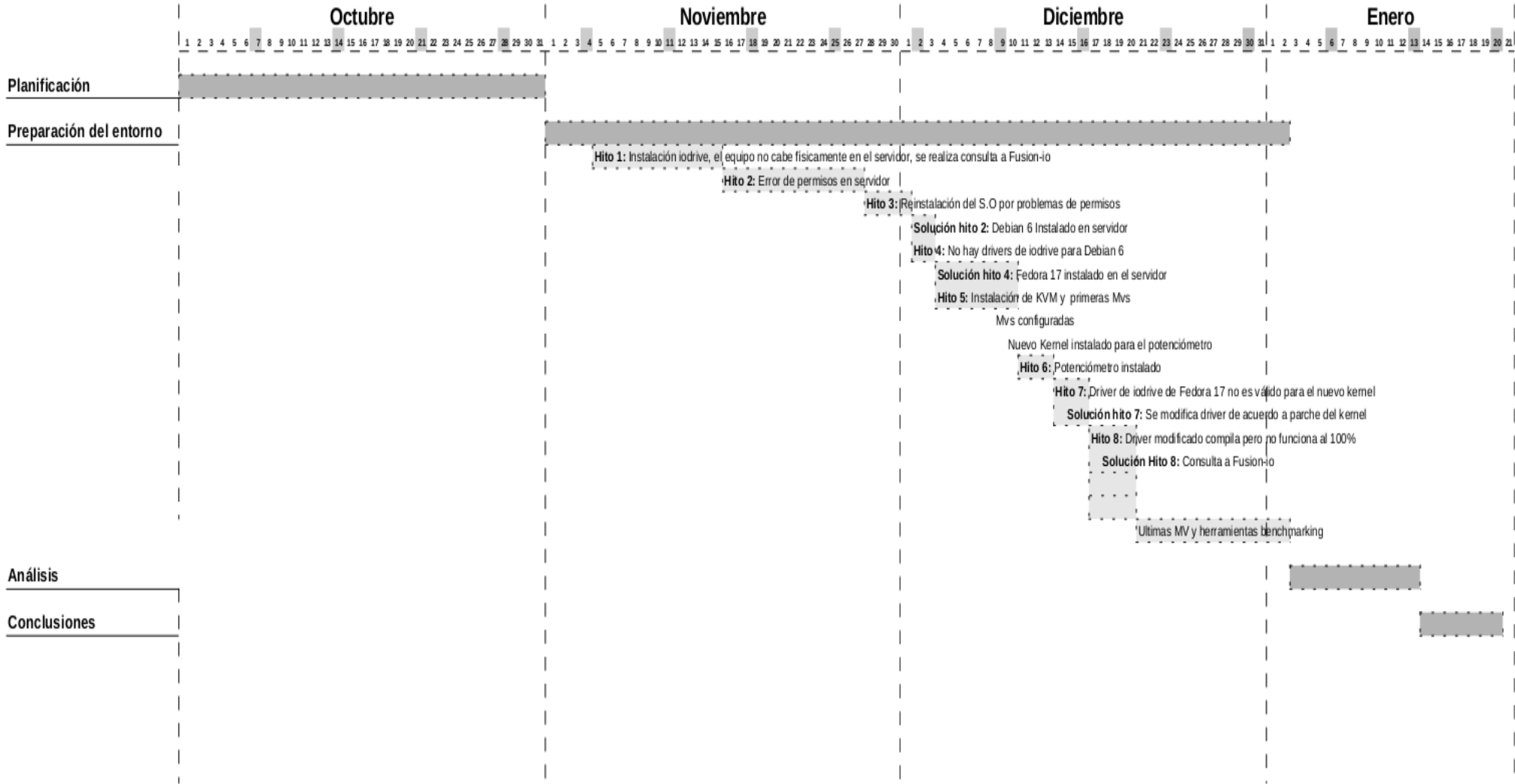
Análisis: Con los datos obtenidos de la etapa anterior realizamos un análisis que nos permitió comparar los costes energéticos de la virtualización en base al hardware empleado.

Conclusiones: A partir del análisis anterior se exponemos las conclusiones obtenidas y presentamos algunas sugerencias para futuros experimentos.

Seguimiento: El seguimiento del proyecto junto con el consultor se llevo principalmente vía correo electrónico. Se concreto el envío programado de una comunicación cada viernes; a fin de comentar el avance del proyecto, plantear dudas y definir objetivos. Adicionalmente, siempre que fue necesario se realizaron conferencias de voz para dotar de una comunicación mas fluida al desarrollo del proyecto.

A continuación presentamos la planificación del proyecto, indicando los hitos que han debido solventarse y algunas de las soluciones adoptadas.

1 benchmarking



2. Message passing interface (MPI)

2.1. Introducción

Antes de entrar en detallé sobre la metodología a seguir es importante definir claramente los elementos que conforman este proyecto. En este apartado se hará un breve descripción del estándar de paso de mensajes MPI para: establecer claramente la importancia de este estándar, tanto en sistemas distribuidos como en computadores paralelos de memoria compartida, y presentar una breve aproximación a su funcionamiento.

Pese a la existencia de una gran variedad de sistemas de comunicación entre procesos el MPI Forum, un grupo de 80 personas pertenecientes a mas de 40 organizaciones, asumió en los años 90 la tarea de desarrollar un estándar. En el desarrollo de este estándar se trato de adoptar las características mas atractivas de los sistemas existentes y se identificaron puntos críticos que supusieron la inclusión de nuevas características. Una vez finalizada la primera versión del estándar de MPI, la 1.1, este fue ampliamente aceptado. A fecha de este proyecto la última versión aprobada es la 3.0.

Entre las ventajas del estándar MPI podemos destacar: en primer lugar su portabilidad, ya que a través de un conjunto de librerías permite su ejecución en máquinas con arquitecturas muy diferentes; es posible por ejemplo, ejecutar un mismo código en diferentes SO. El modelo computacional proporcionado ofusca las características particulares de cada sistema, permitiendo al usuario desentenderse de la arquitectura ya que la implementación realizará todas las acciones necesarias.

Otra ventaja es que su diseño se ha pensado para ofrecer la implementación mas eficiente, y ha demostrado su mejor rendimiento frente a sistemas propietarios. Uno de los motivos es que MPI únicamente especifica el funcionamiento lógico de un operación, evitando el como esa operación se desarrolla. De esta manera puede ser fácilmente implementado en diferentes sistemas, adoptando los beneficios específicos de máquinas diferentes. Su eficiencia reside, además, en características como: evitar el uso de grandes cantidades de información con cada mensaje, favorecer el reuso de computaciones previas y evitar computación extra que pueda afectar al rendimiento.

Una de las características mas deseables para el procesamiento paralelo es la escalabilidad, ya que en este tipo de sistemas es importante la capacidad para ampliar y reducir sus recursos de manera sencilla. En este ámbito, MPI proporciona escalabilidad y soporte en todos sus desarrollos, por ejemplo, gracias a la posibilidad de crear subgrupos de procesos para la comunicación colectiva. De esta manera las aplicaciones pueden crear subgrupos de procesos, limitando los procesos involucrados en cada comunicación.

Por último, es importante que el entorno de paso de mensajes, en que nos manejamos tenga un comportamiento mínimamente conocido y controlable; pues alivia parte de la preocupación ante posibles imprevistos. Esto es así en MPI y podemos decir que esta es

otra de sus ventajas cuantitativas frente a otros.

Este estándar posee algunos términos y convenciones que conviene conocer antes de entrar a detallar su uso. En primer lugar los argumentos de las funciones vienen marcados con las palabras inglesas IN, OUT e INOUT:

IN → La llamada emplea pero no actualiza el argumento marcado.

OUT → La llamada puede actualizar el argumento así marcado.

INOUT → La llamada usa y actualiza el argumento.

Existe un caso de uso especial, cuando se emplean un tipo de dato específico de MPI conocido como *handles*. Este es una suerte de puntero, de tal manera, que si el objeto al que apunta es modificado, esta variable se encontrará marcada con OUT aunque el puntero en si mismo no haya sido modificado.

Los *handles* son estructuras que se emplean para el acceso a un tipo de dato especial conocido como objetos opacos. Los objetos opacos son representaciones de objetos en memoria que no son directamente accesibles por el usuario, que incluso desconoce su tamaño y su estructura.

2.2. Comunicación punto a punto

El funcionamiento básico de MPI se basa en la transmisión de datos entre dos extremos, uno retransmitiendo y el otro recibiendo, a este tipo de comunicación se la conoce como transmisión punto a punto. Su funcionamiento reposa en un *set* completo de funciones de envío y recepción. Estas funciones sirven tanto para manejar la comunicación, como para administrar los recursos necesarios para llevarla a cabo.

La función de envío especifica: un *buffer* de envío en la memoria del emisor, la localización, el tamaño y el tipo. Además, la función de envío especifica un conjunto de campos², en la nomenclatura de MPI conocidos como *envelope* o *message envelope*, que pueden ser usados por el receptor para filtrar los mensajes recibidos y seleccionar un mensaje en particular.

La función de recepción, cuando recibe un mensaje, lo selecciona de acuerdo a la información especificada en su *envelope* y lo almacena en el *buffer* de recepción. Los tres primeros parámetros de la llamada de recepción indican la localización, el tamaño y el tipo del *buffer* de recepción, los tres siguientes son empleados para seleccionar el mensaje entrante y el último parámetro se emplea para devolver información al emisor.

Estas llamadas de comunicación punto a punto pueden ser catalogadas como: bloqueantes y no bloqueantes. El primer tipo implica que el envío se bloquea hasta que el *buffer* de recepción pueda ser usado en exclusiva, igualmente la llamada de recepción se bloquea

² *source, destination, tag y communicator*

hasta que el *buffer* de recepción contenga el mensaje. Las comunicaciones no bloqueantes permiten: superponer el envío de mensajes con la computación, o el envío simultáneo de mensajes. El estándar posee funciones diferentes para unos y otros tipos de llamadas, esto es necesario, pues se necesita manejar parámetros diferentes en cada caso.

La semántica de ambos tipos de comunicación viene determinada por el protocolo subyacente en cada arquitectura. Para lograr estandarizar todas estas semánticas MPI ha incluido todas las opciones más importantes de cada una de ellas, permitiendo que tanto las comunicaciones bloqueantes como las no bloqueantes dispongan de diferentes modos que permiten elegir una semántica u otra.

Los modos disponibles son cuatro: *standard*, donde la finalización del envío no presupone que haya comenzado su recepción; *buffered*, que asegura cierta cantidad de espacio disponible en el *buffer* de recepción; *synchronous*, que para dar mayor seguridad usa una semántica pactada entre emisor y receptor; y por último el modo *ready*, que permite al programador explotar un conocimiento adicional, pues se asegura que la recepción ya ha comenzado y de este modo se adapta al medio.

a) Operaciones de envío bloqueantes

A continuación se presenta la sintaxis de la llamada de envío bloqueante, tanto su descripción general y la de sus argumentos como su sintaxis en lenguaje C y Fortran:

MPI_SEND(buf, count, datatype, dest, tag, comm)

<i>IN buf</i>	<i>Dirección inicial del buffer de envío (choice)</i>
<i>IN count</i>	<i>Numero de elementos en el buffer de envío (entero positivo)</i>
<i>IN datatype</i>	<i>Tipo de dato de cada elemento en el buffer (handle)</i>
<i>IN dest</i>	<i>Destino (entero)</i>
<i>IN tag</i>	<i>Etiqueta del mensaje (entero)</i>
<i>IN comm</i>	<i>Comunicador (handle)</i>

int MPI_Send(void buf, int count, MPI_Datatype datatype, int dest, int tag, MPI_Comm comm)*

MPI_SEND(BUF, COUNT, DATATYPE, DEST, TAG, COMM, IERROR)
<type> BUF()*
INTEGER COUNT, DATATYPE, DEST, TAG, COMM, IERROR

El tamaño del *buffer* de envío se establece como el número *count* de entradas del tipo especificado con la opción *datatype*, comenzando con la entrada especificada en la dirección de memoria *buf*. En MPI la longitud del mensaje viene expresada en número de elementos y no en número de bytes de modo que su medida es independiente de la

arquitectura, y en cada una de ellas se reservara el espacio necesario para almacenar el número necesario de objetos del tipo pasado. Como vemos, por lo tanto, los datos del mensaje consisten en una secuencia de valores de cada tipo indicado por *datatype*. El valor de *count* podría ser cero en cuyo caso el mensaje no contendrá datos. Los tipos de datos que pueden ser especificados corresponderán a los tipos básicos del lenguaje empleado.

Para posibilitar el uso de estas funciones en todas las arquitecturas posibles se hace necesario emplear datos tipados. Esto se consigue asociando una etiqueta *tag* a los datos, de manera que, en cada diferente arquitectura se realicen los cambios necesarios para transformar la información recibida en el tipo de dato indicado. Esta etiqueta permite además, realizar selecciones y filtros por tipo de dato, tanto en el envío como en la recepción.

El argumento *comm*, es un objeto que especifica un dominio de comunicaciones, y nos da el contexto de comunicación en el que se realiza el envío. Este dominio de comunicaciones se trata de una estructura global que permite comunicarse a los procesos de un grupo entre si o con procesos de otro grupo.

La comunicación dentro de un grupo es conocida como intracomunicación³ y la comunicación entre grupos como intercomunicación⁴. Los procesos dentro de cada grupo están ordenados e identificados por su rango⁵ comenzando por el rango cero, perteneciente al proceso *root* o raíz. Cada proceso puede participar de diferentes dominios de comunicación, no están limitados a uno solo, de manera que distintos dominios pueden tener parcial o completamente solapados los procesos que lo conforman. Las aplicaciones MPI comienzan con un dominio de comunicación por defecto que incluye todos los procesos *MPI_COMM_WORLD*.

Como ya se ha indicado ,MPI dispone de diferentes modos que le permiten ajustar su semántica a diferentes necesidades. Para ello dispone de llamadas de envío específicas para los modos *buffered*, *synchronous* y *ready* que se especifican a continuación:

MPI_BSEND(buf, count, datatype, dest, tag, comm)

<i>IN buf</i>	<i>Dirección inicial del buffer de envío (choice)</i>
<i>IN count</i>	<i>Numero de elementos en el buffer de envío (entero positivo)</i>
<i>IN datatype</i>	<i>Tipo de dato de cada elemento en el buffer (handle)</i>
<i>IN dest</i>	<i>Destino (entero)</i>
<i>IN tag</i>	<i>Etiqueta del mensaje (entero)</i>
<i>IN comm</i>	<i>Comunicador (handle)</i>

3 intracommunicator
4 intercommunicator
5 rank

```
int MPI_Bsend(void * buf, int count, MPI_Datatype datatype, int dest, int tag, MPI_Comm comm)
```

```
MPI_BSEND(BUF, COUNT, DATATYPE, DEST, TAG, COMM, IERROR)
<type> BUF(*)
INTEGER COUNT, DATATYPE, DEST, TAG, COMM, IERROR
```

El modo *buffered* puede comenzar sin haber recibido una confirmación de recepción y puede terminar a pesar de no recibirla. Se trata de una operación local y no depende en ningún caso del receptor. MPI asegurará el envío del mensaje almacenando, el mensaje saliente, para permitir que se complete la transmisión.

MPI_SSEND(buf, count, datatype, dest, tag, comm)

<i>IN buf</i>	<i>Dirección inicial del buffer de envío (choice)</i>
<i>IN count</i>	<i>Numero de elementos en el buffer de envío (entero positivo)</i>
<i>IN datatype</i>	<i>Tipo de dato de cada elemento en el buffer (handle)</i>
<i>IN dest</i>	<i>Destino (entero)</i>
<i>IN tag</i>	<i>Etiqueta del mensaje (entero)</i>
<i>IN comm</i>	<i>Comunicador (handle)</i>

```
int MPI_SSend(void * buf, int count, MPI_Datatype datatype, int dest, int tag, MPI_Comm comm)
```

```
MPI_SSEND(BUF, COUNT, DATATYPE, DEST, TAG, COMM, IERROR)
<type> BUF(*)
INTEGER COUNT, DATATYPE, DEST, TAG, COMM, IERROR
```

En el modo *synchronous* el envío puede comenzar sin confirmación de recepción, pero solo puede finalizar si se recibe la confirmación de recepción. Por lo tanto, la finalización de la llamada de envío, además de indicar que el *buffer* de envío esta libre, indica que ha comenzado la recepción.

MPI_RSEND(buf, count, datatype, dest, tag, comm)

<i>IN buf</i>	<i>Dirección inicial del buffer de envío (choice)</i>
<i>IN count</i>	<i>Numero de elementos en el buffer de envío (entero positivo)</i>
<i>IN datatype</i>	<i>Tipo de dato de cada elemento en el buffer (handle)</i>
<i>IN dest</i>	<i>Destino (entero)</i>
<i>IN tag</i>	<i>Etiqueta del mensaje (entero)</i>
<i>IN comm</i>	<i>Comunicador (handle)</i>

```
int MPI_RSend(void * buf, int count, MPI_Datatype datatype, int dest, int tag,
```

MPI_Comm comm)

MPI_RSEND(BUF, COUNT, DATATYPE, DEST, TAG, COMM, IERROR)

<type> BUF()*

INTEGER COUNT, DATATYPE, DEST, TAG, COMM, IERROR

En el modo *ready* el envío solo comenzará si tenemos confirmación de la recepción. Esto se consigue enviando una notificación al receptor, de tal manera, que solamente se iniciará la comunicación si se recibe una notificación, de respuesta, informando que el receptor esta preparado para recibir.

b) Operaciones de recepción bloqueantes

La sintaxis de la llamada de recepción bloqueante es:

MPI_RECV (buf, count, datatype, source, tag, comm, status)

<i>OUT buf</i>	<i>Dirección inicial del buffer de recepción (choice)</i>
<i>IN count</i>	<i>Número de elementos en el buffer de recepción</i>
<i>IN datatype</i>	<i>Tipo de dato de cada elemento en el buffer (handle)</i>
<i>IN source</i>	<i>Rango⁶ de la fuente (entero)</i>
<i>IN tag</i>	<i>Etiqueta del mensaje (entero)</i>
<i>IN comm</i>	<i>Comunicador (hanle)</i>
<i>IN status</i>	<i>Estatus (Status)</i>

int MPI_Recv(void, buf, int, count, MPI_Datatype datatype, int source, int tag, MPI_Comm comm, MPI_Status *status)*

MPI_RECV(BUF, COUNT, DATATYPE, SOURCE, TAG, COMM, STATUS, IERROR)

<type> BUF()*

INTEGER COUNT, DATATYPE, SOURCE, TAG, COMM, STATUS(MPI_STATUS_SIZE), IERROR

Al igual que en la llamada de envío, el *buffer* de recepción consiste en *count* elementos consecutivos del tipo *datatype* comenzando en la dirección *buf* y el espacio que ocupen vendrá determinado por la arquitectura en la que nos encontremos. En cualquier caso, la longitud del *buffer* debe ser mayor o igual a la longitud del mensaje. En caso de no ser así se produciría un error por desbordamiento si los datos no caben en el *buffer*.

La operación de recepción esta regida por los valores especificados en el *message envelope* en la llamada de envío. Para que comience la recepción, los valores de

6 rank

source, *tag* y *comm* deben coincidir con los especificados en la llamada de recepción. Existe la opción de aceptar cualquier etiqueta, especificando en el campo *tag* el valor *MPI_ANY_TAG*. También se dispone de una opción, para no especificar la fuente con el valor *MPI_ANY_SOURCE* en este caso en el campo *source*. No es posible, sin embargo, aceptar un valor cualquiera para el comunicador, este deberá coincidir con el especificado en el receptor.

EL argumento *status* de la llamada de recepción es empleado para devolver información. *Status* corresponde a un tipo de datos definido en MPI que se corresponde a una suerte de estructura formada por tres campos en lenguaje C y un vector de enteros en Fortran. Este tipo de estructuras deben ser declarados por el usuario dado que no son objetos del sistema.

c) Comparación y conversión de tipos de datos

La comparación de datos en MPI es un punto problemático debido a su capacidad multiplataforma y al empleo de *tags*. Por lo tanto debe ser fuertemente observado principalmente en tres pasos:

- Los datos se sacan del *buffer* de envío y se ensamblan
- Un mensaje se transfiere del emisor al receptor
- Los datos se desensamblan en el *buffer* de recepción

El tipo especificado para cada variable en el *buffer* de envío deberá corresponderse con la variable declarada en la notificación de envío, la cual a su vez, deberá coincidir con el tipo especificado en la operación de recepción y esta, una vez mas, con el *buffer* de recepción.

Para especificar los tipos de datos MPI dispone de sus propios tipos de datos declarados *MPI_INTEGER*, *MPI_REAL*, *MPI_CHARACTER*.... etc. Estos tipos, a su vez, se encuentran relacionadas en una tabla con los tipos definidos en el lenguaje huésped de tal manera que por ejemplo *MPI_INTEGER* corresponderá a a un tipo *INTEGER* de Fortran.

Existen dos excepciones a esto: *MPI_PACKED*, que se emplea para el envío de datos empaquetados; y *MPI_BYTE*, que permite transferir valores binarios de un byte, en memoria, sin modificarlos. La reglas de comparación de tipos de pueden clasificar por lo tanto en tres grupos:

- Comunicación de datos tipados.
- Comunicación de datos no tipados al usar *MPI_BYTE*.
- Comunicación de datos empaquetados cuando se emplea *MPI_PACKED*.

d) Comunicaciones no bloqueantes

Las operaciones no bloqueantes pueden ser de gran utilidad para mejorar el rendimiento de un sistema solapando comunicación y computación, si se dispone del hardware adecuado. Se dispone de una llamada *send start* que comienza la comunicación, pero que no la finaliza; y de una llamada *send complete* que finaliza la comunicación. Análogamente, disponemos de la llamada *receive start*, que inicia la operación de recepción pero no la completa; y de una llamada *receive complete*, que verifica que los datos se han copiado en el *buffer* receptor. Con las llamadas no bloqueantes el programador no sabe si los datos transmitidos han sido copiados en el destino y se ve obligado a comprobar el estado del *buffer* si quiere volver a usarlo.

Las operaciones de envío no bloqueantes disponen de los mismos cuatro modos que las operaciones bloqueantes, y a excepción del modo *ready*, el envío puede comenzar aunque no se tenga confirmación de la recepción. En todos los casos *send start* funciona como una llamada local y finaliza inmediatamente, la llamada *send complete* finalizará cuando los datos del *buffer* de envío hayan sido copiados.

En modo *synchronous*, la llamada *send complete* no es local y solo finaliza si esta asegurada la recepción del mensaje. La comunicación será completa si la llamada de recepción asociada se produce antes de que se produzca la llamada *receive complete*. Si el modo es *buffered*, entonces el mensaje debe ser enviado si no existen recepciones pendientes, en este caso, la llamada *send complete* es local y sucederá independientemente del estado de la recepción.

Las funciones de envío no bloqueantes pueden ser relacionadas con funciones de llamadas de recepción bloqueantes y viceversa.

La comunicación no bloqueante emplea un objeto opaco *request* para identificar la operación que comienza la comunicación, y relacionarla con la comunicación que la termina. Estos objetos de sistema *request* son accesibles a través de *handles*. Los *request* especifican varias propiedades de una operación de comunicación: el modo de envío, el *buffer* de comunicación asociado, su contexto, la etiqueta y el destino. Además, también almacena información acerca del estatus de la operación de comunicación pendiente.

La estructura de la llamada no bloqueante es similar a la de la bloqueante pero añadiendo el prefijo I, del inglés *immediate*, e igualmente usamos letras para identificar cada uno de los modos: B para el modo *buffered*, la S para *synchronous* y la R para *ready*.

MPI_ISEND(buf, count, datatype, dest, tag, comm, request)

IN buf

Dirección inicial del buffer de envío (choice)

<i>IN count</i>	<i>Numero de elementos en el buffer de envío (entero positivo)</i>
<i>IN datatype</i>	<i>Tipo de dato de cada elemento en el buffer (handle)</i>
<i>IN dest</i>	<i>Destino (entero)</i>
<i>IN tag</i>	<i>Etiqueta del mensaje (entero)</i>
<i>IN comm</i>	<i>Comunicador (handle)</i>
<i>OUT request</i>	<i>Objeto request (handle)</i>

int MPI_Isend(void buf, int count, MPI_Datatype datatype, int dest, int tag, MPI_Comm comm, MPI_Request *request)*

MPI_ISEND(BUF,COUNT,DATATYPE,DEST, TAG, COMM, REQUEST, IERROR)
<type> BUF()*
INTEGER COUNT, DATATYPE, DEST, TAG, COMM, REQUEST, IERROR

Análogamente la llamada no bloqueante de recepción es:

MPI_Irecv(buf, count, datatype, source, tag, comm, request)

<i>OUT buf</i>	<i>Dirección inicial del buffer de recepción (choice)</i>
<i>IN count</i>	<i>Número de elementos en el buffer de recepción</i>
<i>IN datatype</i>	<i>Tipo de dato de cada elemento en el buffer (handle)</i>
<i>IN source</i>	<i>Rango⁷ de la fuente (entero)</i>
<i>IN tag</i>	<i>Etiqueta del mensaje (entero)</i>
<i>IN comm</i>	<i>Comunicador (handle)</i>
<i>OUT request</i>	<i>Puntero al objeto request (handle)</i>

int MPI_Irecv(void, buf, int, count, MPI_Datatype datatype, int source, int tag, MPI_Comm comm, MPI_Request *request)*

MPI_Irecv(BUF, COUNT, DATATYPE, SOURCE, TAG, COMM, REQUEST, IERROR)
<type> BUF()*
INTEGER COUNT, DATATYPE, SOURCE, TAG, COMM, REQUEST, IERROR

Estas llamadas asignan un objeto *request* que es referenciado por el *handle request* pasado como argumento en la llamada. Este objeto puede ser consultado luego para conocer cual es el estado de la comunicación.

Una llamada no bloqueante de envío indica que van a comenzar a copiarse los datos fuera del *buffer* de emisión y termina, por lo que hasta que el envío no este completo no debería accederse al *buffer*. En la recepción las implicaciones son exactamente las mismas, pero en este caso, al copiarse los datos en el *buffer* de recepción.

⁷ rank

Para completar la comunicación las llamadas no bloqueantes disponen de dos funciones adicionales *MPI_WAIT* y *MPI_TEST*. La completación de una comunicación indica, que el emisor es libre de actualizar el *buffer* de envío, si bien, no indica que la comunicación haya sido recibida. En el caso del modo *synchronous* la finalización indica que una llamada análoga de recepción ha comenzado.

La llamada *MPI_WAIT* finaliza cuando la operación especificada en el objeto *request*, pasado como argumento, se ha completado. La llamada devuelve en el objeto *status* información acerca de la comunicación. La sintaxis de la llamada *MPI_WAIT* es la siguiente:

MPI_WAIT(request, status)

<i>INOUT request</i>	<i>Puntero al objeto request (handle)</i>
<i>OUT status</i>	<i>Objeto status (Status)</i>

*int MPI_Wait(MPI_Request *request, MPI_Status *status)*

MPI_WAIT(REQUEST, STATUS, IERROR)

INTEGER REQUEST, STATUS(MPI_STATUS_SIZE), IERROR

La llamada *MPI_TEST* devuelve *flag* = cierto si la operación identificada por el objeto *request* se ha completado. En ese caso, el objeto *status* contendrá información referente a la completación de la operación, en caso contrario su valor es falso por no haberse completado la operación y el valor del objeto *status* no se encontrará definido. La sintaxis de esta llamada es:

MPI_TEST(request, flag, status)

<i>INOUT request</i>	<i>Puntero al objeto request (handle)</i>
<i>OUT flag</i>	<i>cierto si la operación se ha completado (lógico)</i>
<i>OUT status</i>	<i>Objeto status (Status)</i>

*int MPI_Test(MPI_Request *request, int *flag, MPI_Status *status)*

MPI_TEST(REQUEST, FLAG, STATUS, IERROR)

LOGICAL FLAG

INTEGER REQUEST, STATUS(MPI_STATUS_SIZE), IERROR

2.3. **Comunicaciones colectivas**

MPI proporciona diversas funciones para las comunicaciones colectivas. El término comunicaciones colectivas implica aquellas comunicaciones que se producen dentro de un grupo de procesos, una llamada colectiva se produce cuando todos los procesos del grupo

comunicaciones colectivas implica aquellas comunicaciones que se producen dentro de un grupo de procesos, una llamada colectiva se produce cuando todos los procesos del grupo llaman a la rutina de comunicación con los mismos argumentos. La semántica y sintaxis de estas llamadas está pensada para ser consistente con la de las comunicaciones punto a punto.

Uno de los argumentos de este tipo de llamadas es un objeto *communicator* que define el grupo al que pertenecen los procesos participantes y proporciona un contexto para la comunicación. Este objeto *communicator* es el mismo que el empleado en las comunicaciones punto a punto, pero el estándar garantiza que no será posible confundir un mensaje generado en una comunicación punto a punto con otro mensaje de carácter colectivo. El *communicator* deberá ser un intracomunicador, pues los intercomunicadores que unen mas de un grupo no están permitidos.

Cuando el origen o destino de una comunicación colectiva es un único proceso este proceso es conocido como *root*. Algunos argumentos de las llamadas colectivas solo son empleados si en la comunicación se encuentra implicado uno de estos procesos. En base a esto, las comunicaciones colectivas pueden dividirse en tres tipos: *root* manda datos a todos los procesos, *broadcast* y *scatter*; *root* recibe datos de todos los procesos, *gather*; y cada proceso se comunica con cada proceso; *allgather*, *alltoall*.

Las comunicaciones colectivas comparativamente son menos versátiles que las comunicaciones punto a punto, esto es así por una decisión de diseño que ha permitido mantener los niveles de complejidad de su sintaxis y semántica en unos límites razonables. Por ejemplo, para que se de una comunicación colectiva el número de datos enviados ha de coincidir exactamente con la cantidad especificada por el receptor, y además, las condiciones de comparación de tipos son mas estrictas. Para mantener esta baja complejidad, en sus versiones iniciales MPI no permitía comunicaciones colectivas no bloqueantes, sin embargo esto ya es posible en su versión 3.0.

Para ejecutar una comunicación colectiva todos los procesos de un grupo deben invocar la rutina de comunicación con los mismos argumentos. Así, las comunicaciones colectivas no usan un argumento *tag*, sino que son estrictamente emparejadas siguiendo el orden de ejecución y no disponen tampoco de modos. Se puede decir que su comportamiento es análogo al modo estándar de las comunicaciones punto a punto.

Una función colectiva retorna en cuanto su participación en la comunicación ha finalizado. Al igual que en una comunicación no bloqueante, la finalización indica que es posible acceder al *buffer* de comunicación, pero no indica que otros procesos hayan finalizado o comenzado su operación. Así una operación colectiva puede o no sincronizar todos los procesos llamantes según la descripción del estándar, y existen implementaciones que ofrecen ambas funcionalidades.

a) Broadcast

Esta función emite un mensaje desde el proceso *root* a todos los procesos del grupo. El argumento *root* deberá ser igual en todos los procesos y *comm* debe representar el mismo dominio de comunicación intragrupo. Cuando la función retorna, el contenido del *buffer* de comunicación del proceso *root* se habrá copiado en todos los procesos del grupo. Los tipos de datos derivados están permitidos para *datatype* pero los tipos de *count* y *datatype* de cualquier proceso deben ser iguales a los del proceso *root*. Esto implica que los datos enviados y recibidos serán exactamente los mismos.

MPI_BCAST(buffer, count, datatype, root, comm)

<i>INOUT buffer</i>	<i>Dirección donde comienza el buffer.</i>
<i>IN count</i>	<i>Número de entradas en el buffer.</i>
<i>IN datatype</i>	<i>Tipos de datos del buffer.</i>
<i>IN root</i>	<i>Rango del proceso root.</i>
<i>IN comm</i>	<i>Objeto communicator</i>

int MPI_Bcast(void buffer, int count, MPI_Datatype datatype, int root, MPI_Comm comm)*

MPI_BCAST(BUFFER, COUNT, DATATYPE, ROOT, COMM, IERROR)
<type> BUFFER()*
INTEGER COUNT, DATATYPE, ROOT, COMM, IERROR

b) Gather

En este caso, todos los procesos del grupo envían el contenido de su *buffer* al proceso *root*. Este recibirá todos los mensajes y los almacenará ordenados por orden de según el rango del proceso que lo envía. Una descripción válida sería decir que los mensajes enviados por los n-1 procesos del grupo, donde n son todos los procesos que lo componen, son concatenados en un único mensaje ordenado por rango que es recibido por el proceso *root*. Durante su ejecución el *buffer* de recepción es ignorado por todos los procesos emisores. Los tipos derivados también están permitidos y sus restricciones en este sentido son las mismas que las indicadas para *broadcast*.

MPI_GATHER(sendbuf, sendcount, sendtype, recvbuf, recvcount, recvtype, root, comm)

<i>IN sendbuf</i>	<i>Dirección de comienzo del buffer de envío.</i>
<i>IN sendcount</i>	<i>Número de elementos en el buffer de envío.</i>
<i>IN sendtype</i>	<i>Tipo de datos de los elementos en el buffer de envío.</i>
<i>OUT recvbuf</i>	<i>Dirección del buffer de recepción.</i>
<i>IN recvcount</i>	<i>Número de elementos para cada receptor.</i>
<i>IN recvtype</i>	<i>Tipo de datos de los elementos del buffer de recepción.</i>
<i>IN root</i>	<i>Rango dentro del grupo del proceso root.</i>
<i>IN comm</i>	<i>Objeto communicator.</i>

```
int MPI_Gather(void * sendbuf, int sendcount, MPI_Datatype sendtype, void*
    recvbuf, int recvcount, MPI_Datatype recvtype, int root,
    MPI_Comm comm)
```

```
MPI_GATHER(SENDBUF, SENDCOUNT, SENDTYPE, RECVBUF,
    RECVCOUNT, RECVTYPE, ROOT, COMM, IERROR)
<type> SENDBUF(*), RECVBUF(*)
INTEGER SENDCOUNT, SENDTYPE, RECVCOUNT, RECVTYPE, ROOT, COMM, IERROR
```

c) Scatter

Es la operación inversa de *Gather*, es decir, el proceso *root* envía el contenido de su *buffer* de envío, consistente en un vector de elementos, al de todos los restantes procesos del grupo definido por el *communicator*. Es, por lo tanto, el equivalente a que el proceso *root* efectuó $n-1$ operaciones de envío *MPI_Send*, donde n es el número total de procesos del grupo, y cada uno de los procesos efectuó una operación de recepción *MPI_Recv*. Al igual que en los casos anteriores las restricciones de tipo deben ser respetadas y el número de datos enviados debe ser igual al número de datos recibidos. En todos los procesos receptores los argumentos *root* y *comm* deberán ser iguales y el argumento *sendbuffer* será ignorado.

MPI_SCATTER(sendbuf, sendcount, sendtype, recvbuf, recvcount, recvtype, root, comm)

<i>IN sendbuf</i>	<i>Dirección del buffer de envío.</i>
<i>IN sendcount</i>	<i>Número de elementos mandados a cada proceso.</i>
<i>IN sendtype</i>	<i>Tipo de datos de los elementos enviados.</i>
<i>OUT recvbuf</i>	<i>Dirección del buffer de recepción.</i>
<i>IN recvcount</i>	<i>Número de elementos en el buffer de recepción.</i>
<i>IN root</i>	<i>Rango del proceso root.</i>
<i>IN comm</i>	<i>Objeto communicator.</i>

```
int MPI_Scatter(void* sendbuf, int sendcount, MPI_Datatype sendtype, void *
    recvbuf, int recvcount, MPI_Datatype recvtype, int root,
    MPI_Comm comm)
```

```
MPI_SCATTER(SENDBUF, SENDCOUNT, SENDTYPE, RECVBUF,
    RECVCOUNT, RECVTYPE, ROOT, COMM, IERROR)
<type> SENDBUF(*), RECVBUF(*)
INTEGER SENDCOUNT, SENDTYPE, RECVCOUNT, RECVTYPE, ROOT, COMM,
IERROR
```

d) Gather to all

Puede plantearse de la misma manera que *Gather* pero en este caso son todos los procesos los que reciben el mensaje además de *root*. El enésimo bloque de datos

enviado por cada proceso es recibido, por cada proceso, en el *enésimo* bloque del *buffer* de recepción. El efecto es el mismo que si cada proceso ejecutara *n* llamadas a *MPI_Gather* donde *root = 0, ..., n-1*. Las reglas de uso por lo tanto son las mismas que las aplicables para *MPI_Gather*

MPI_ALLGATHER(sendbuffer, sendcount, sendtype, recvbuf, recvcount, recvtype, comm)

<i>IN sendbuf</i>	<i>Dirección del buffer de envío.</i>
<i>IN sendcount</i>	<i>Número de elementos en el buffer de envío.</i>
<i>IN sendtype</i>	<i>Tipo de datos de los elementos en el buffer.</i>
<i>OUT recvbuf</i>	<i>Dirección del buffer de recepción.</i>
<i>IN recvcount</i>	<i>Número de elementos recibidos de cualquier proceso.</i>
<i>IN recvtype</i>	<i>Tipo de datos de los elementos recibidos.</i>
<i>IN comm</i>	<i>Objeto communicator.</i>

```
int MPI_Allgather(void* sendbuf, int sendcount, MPI_Datatype sendtype, void*  
recvbuf, int recvcount, MPI_Datatype recvtype, MPI_Comm  
comm)
```

```
MPI_ALLGATHER(SENDBUF, SENDCOUNT, SENDTYPE, RECVBUF, RECVCOUNT,  
RECVTYPE, COMM, IERROR)  
<type>SENDBUF(*), RECVBUF(*)  
ININTEGER SENDCOUNT, SENDTYPE, RECVCOUNT, RECVTYPE, COMM, IERROR
```

e) All to All Scatter/Gather

Se trata de una extensión de la función *MPI_ALLGATHER* para los casos en los que sea necesario que cada proceso emisor envíe datos diferentes a cada proceso receptor. El *enésimo* bloque de datos enviado por el proceso *i* es recibido por el proceso *n* y colocado en el *iesimo* bloque del *buffer* de recepción. Las limitaciones en cuanto a tipos de datos y cantidad de datos enviados y recibidos son las mismas que en las funciones anteriores.

MPI_ALLTOALL(sendbuf, sendcount, sendtype, recvbuf, recvcount, recvtype, comm)

<i>IN sendbuf</i>	<i>Dirección de comienzo del buffer de envío.</i>
-------------------	---

<i>IN sendcount</i>	Número de elementos en el buffer de envío.
<i>IN sendtype</i>	Tipo de los elementos en el buffer de envío.
<i>OUT recvbuf</i>	Dirección de l buffer de recepción.
<i>IN recvcount</i>	Número de elemento recibidos de cualquier proceso.
<i>IN recvtype</i>	Tipo de los elementos en el buffer de recepción.
<i>IN comm</i>	Objeto <i>communicator</i> .

```
int MPI_Alltoall(void* sendbuf, int sendcount, MPI_Datatype sendtype, void*
recvbuf, int recvcount, MPI_Datatype recvtype, MPI_Comm
comm)
```

```
MPIALLTOALL(SENDBUF,SENDcount,SENDTYPE,RECVBUF,
RECVcount,RECVTYPE,COMM,IERROR)
```

```
<type>SENDBUF(*),RECVBUF(*)
```

```
INTEGER SENDCOUNT, SENDTYPE, RECVCOUNT,RECVTYPE,COMM,IERROR
```

Para cada una de las funciones presentadas, a excepción de *Broadcast*, MPI ofrece también una función alternativa, que permite el envío de una cantidad variada de datos para cada proceso. Para ello, el argumento *recvcount* y *sendcount*, pasan de ser un entero a ser un vector de enteros. Se añade además un nuevo argumento *displs* que es también un vector de enteros y que indica los desplazamientos que deben hacerse, para que los datos sean ordenados correctamente. El nombre de estas funciones es el mismo que el de las ya descritas pero añadiendo una V de vector al final, por ejemplo, *MPI_GATHERV*.

2.4. Los objetos *communicator*

En sistemas distribuidos resulta esencial que distintos grupos de procesos sean capaces de trabajar independientemente para realizar tareas distintas. En MPI esto se consigue gracias a los objetos *communicator*, en español comunicadores. Estos grupos son conjuntos ordenados de procesos identificados por un rango. El rango son números enteros consecutivos comenzando en el 0. El estándar dispone de un grupo predefinido, *MPI_GROUP_EMPTY*, que es un grupo sin ningún proceso miembro. MPI proporciona diferentes operaciones para el manejo y creación de grupos, esto es útil por ejemplo, cuando una aplicación tiene la información que precisa distribuida en varios nodos, y gracias a estas operaciones puede crear un nuevo grupo que los englobe.

Los objetos *communicator* se encuentran en la gran mayoría de llamadas de MPI, permitiendo simplificar las llamadas a funciones; reduciendo el número de argumentos y favoreciendo la escalabilidad del estándar. Los comunicadores especifican dominios de comunicación, que están compuestos por conjuntos de comunicadores, con valores consistentes uno por cada proceso participante, siendo cada comunicador la representación del dominio de comunicación local.

Los *communicator* son objetos opacos compuestos por atributos y unas reglas que gobiernan su creación y su destrucción. Como ya se ha comentado anteriormente existen dos tipos: los intracomunicadores y los intercomunicadores. Los intracomunicadores son

los comunicadores empleados para la comunicación dentro de un mismo grupo, y los intercomunicadores se emplean para las comunicaciones entre grupos diferentes. Ambos tipos de comunicadores tiene dos atributos, si bien, en el caso de los intracomunicadores se trata del grupo y la topología; y en el de los intercomunicadores son los dos grupos implicados.

En caso de que el dominio de comunicación sea un intragrupo todos sus comunicadores serán intracomunicadores y todos tendrán un atributo grupo idéntico. Un comunicador se puede visualizar como un vector de enlaces con índices consistentes, en los que la n -ésima posición corresponderá al proceso n . En el caso de los intracomunicadores, además, este vector contendría un enlace a si mismo de manera que forme un grafo completo.

El atributo rango en las comunicaciones punto a punto, que especifica un proceso concreto, se entenderá como perteneciente al grupo o grupos referenciados por el comunicador. Paralelamente, en una comunicación colectiva todos los procesos involucrados deberán usar el mismo intracomunicador. Cabe destacar, que aunque por simplicidad cuando dos comunicadores son iguales se dice que son el mismo, realmente se trata de objetos distintos. En las comunicaciones colectivas no esta permitido usar intercomunicadores.

Los intercomunicadores permiten la comunicaciones entre diferentes grupos de procesos. Un intergrupo esta compuesto por un conjunto de intercomunicadores con un par de grupos disjuntos como sus atributos. A diferencia de los intracomunicadores, sus enlaces forman un grafo bipartido donde cada proceso de un grupo esta enlazado con todos los procesos del otro grupo y viceversa. Al igual que en los intracomunicadores, los enlaces son índices consistentes y el n -ésimo enlace en un grupo apunta al proceso de rango n del otro grupo.

Los intercomunicadores diferencian entre grupos locales y grupos remotos, pero emplean la misma lógica que los intracomunicadores usando un par de rangos para la comunicación, así cuando un proceso de un grupo local envía un mensaje, y un proceso parejo análogo en un grupo remoto realizan una recepción, el primer rango se referirá al proceso local y el segundo al proceso remoto.

Las propiedades de los intercomunicadores por lo tanto son: 1) poseer la misma sintaxis que los intracomunicadores, 2) los procesos objetivos son identificados por su rango en el grupo remoto, 3) el estándar garantiza que no habrá conflicto con otras comunicaciones que empleen comunicadores diferentes, 4) no puede usarse para comunicaciones colectivas, y 5) un comunicador puede se intercomunicador o intracomunicador pero no ambos.

MPI proporciona la función `MPI_COMM_TEST_INTER` para saber si un comunicador es un intercomunicador o un intracomunicador.

3. Herramientas

En el transcurso de este proyecto son varias las herramientas tanto de hardware como de software que hemos usado a parte del ya expuesto MPI. Por ello, a continuación se presenta brevemente una introducción a cada una de los elementos que se han visto implicados. La motivación en este caso es plantear una base para la pruebas a realizar y por ello se exponen no solo las opciones de funcionamiento, si no, una breve introducción sobre las características de diseño y las particularidades existentes.

3.1. KVM

La llegada de la tecnología VT de Intel y SVM de AMD facilitó significativamente el desarrollo de hipervisores. Esto propició que la empresa Israeli Qumranet se embarcara en la creación de un proyecto de virtualización llamado KVM. Esta empresa fue más tarde comprada por RedHat, convirtiendo un proyecto que originalmente tenía como objetivo ser una herramienta de virtualización de escritorio para Windows, en una herramienta de virtualización llamada a reemplazar a la existente Xen.

Los modelos de hipervisores existentes en 2006 consistían en una capa de software que multiplexaba el hardware entre diferentes sistemas operativos huésped; y realizaba tareas básicas de organización y manejo de memoria. La enorme complejidad del hardware terminó haciendo esta tarea altamente complicada al verse involucrados sistemas como NUMA⁸. El acierto de Qumranet en este sentido fue darse cuenta de que ya existía un gestor de memoria eficiente desarrollado: el kernel de Linux.

Ya bajo propiedad de RedHat, la incorporación al desarrollo de KVM de empresas como IBM o Suse contribuyó a que finalmente KVM pasara, a partir de la versión 2.6.20, a formar parte de la rama principal del kernel de Linux. De esta manera todos los avances que se realicen en el código de Linux repercuten a su vez en el funcionamiento de KVM, y los ya existentes son empleados para su funcionamiento.

Esta inclusión en el código principal del kernel ha implicado que se encuentre ya por defecto en prácticamente todas las distribuciones Linux, y que no disponga por sí misma de una versión propia. Así, para determinar que versión estamos empleando solamente es necesario ver cuántas nuevas versiones del kernel existen desde la 2.6.20, en el caso que nos ocupa, que empleamos una distribución “*Fedora 17*”.

KVM consiste en una serie de módulos cargables en el kernel, uno para proporcionar soporte de virtualización del núcleo, y otro específico del procesador. Estos módulos permiten a Linux trabajar como un *baremetal hypervisor*⁹, ejecutándose directamente sobre el hardware, y tratando las máquinas virtuales¹⁰ como cualquier otro proceso de Linux. Dado que KVM está integrado en el kernel, emplea todas las herramientas

8 Non uniform memory access

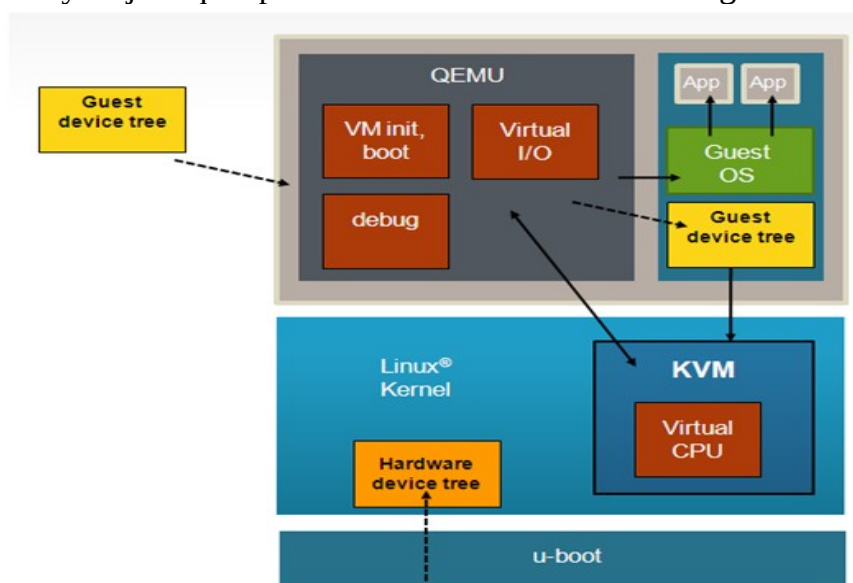
9 Type 1 Hypervisor

10 De aquí en adelante VM

disponibles en este para el manejo de memoria y procesos, evitando que estas funciones tengan que reescribirse con objeto de la virtualización. Además, a los dos modos de ejecución existentes para los procesos, usuario y kernel, añade un tercero llamado *guest mode*, cada uno de estos tres modos tiene asignadas tareas diferentes para la virtualización.

Pero el beneficio no se limita a la gestión de procesos: el almacenamiento dispone de todos los formatos disponibles para Linux y también se aprovechan las medidas de seguridad implementadas como los sandboxes, de manera que las MV¹¹, al funcionar como procesos, están protegidas entre ellas y contra cualquier posible vulnerabilidad del hipervisor. Para proporcionar virtualización de dispositivos KVM emplea una versión modificada de otro proyecto de software libre QEMU, basado en la traducción dinámica de binarios.

La integración de KVM con QEMU puede llevar a confusión. Si bien QEMU es capaz de virtualizar la CPU no dispone de las mismas capacidades que KVM, que trabaja como un driver del kernel. Así KVM, al funcionar como un driver, proporciona eficazmente servicios de CPU a QEMU, a través de un API, y se encarga del manejo de las operaciones que involucren al SO huésped. Por su parte QEMU se encarga de: virtualizar los servicios de E/S, cargar las imágenes necesarias: kernel, árbol de dispositivos .. etc. Finalmente ambos funcionan conjuntamente para proporcionar soporte para la MV. Esta estructura y su jerarquía puede verse claramente en la imagen a continuación.



Descargado de <http://www.freescale.com/>

En cuanto al interfaz se ha tratado de evitar los errores cometidos con Xen que derivaron en un interfaz mal gestionado. Para ello se han estandarizado las librerías *libvirt* y

11 Máquinas virtuales

libguestfs como el API básico para el manejo y creación de las MV con KVM y cualquier aplicación de mayor nivel es construida a partir de estas.

Por último cabe mencionar la OVA (Open Virtualization Alliance) que engloba a todas las empresas involucradas en el desarrollo y promoción de KVM y cuya misión es popularizar su uso, principalmente generando documentación, y encarar los retos futuros en virtualización que pudieran presentarse.

3.2. **Benchmarking**

El termino ingles *benchamking* se emplea para referirse a todas aquellas pruebas que se realizan para obtener el rendimiento de un sistema bajo diferentes situaciones. Estas pruebas se realizan fijando unos parámetros y evaluando otros, y por lo tanto proponen diferentes pruebas para evaluar correctamente un sistema. En el proyecto que nos ocupa hemos elegido tres herramientas para evaluar el rendimiento de las MV's sobre los diferentes tipos de almacenamiento.

Concretamente la intención del proyecto es evaluar este rendimiento en entornos habituales de los actuales centros de datos, es decir: servidores web, servidores de bases de datos y servidores de archivos. Para ello se han seleccionado tres herramientas *httperf*, *sysbench* y *b_eff_io*, respectivamente, como las mas óptimas para realizar nuestras pruebas. A continuación se presenta una breve exposición de sus funcionalidad y características.

a) sysbench

Es una herramienta de benchmarking modular y multiplataforma, escrita por Alexey Kopytov, que permite tener una impresión rápida del rendimiento del sistema. Lo hace a través de diferentes parámetros del SO¹² relativos al funcionamiento en entornos con alta carga. Su funcionamiento es muy sencillo, la aplicación ejecuta un número determinado de hilos que a su vez realizan peticiones en paralelo. La carga producida en el sistema dependerá principalmente del modo elegido.

Permite realizar test en 6 diferentes modos de operación: *cpu*, *threads*, *mutex*, *memory*, *fileio* y *oltp*:

- El modo **cpu** es uno de los mas simples, su funcionamiento se basa en el calculo de números primos. Cada hilo lanzado procederá a calcular los números primos hasta un valor dado por la opción `-cpu-max-primos` empleando enteros de 64 bits.

sysbench - -test=cpu - -cpu-max-prime= "valor a calcular" run

12 Sistema operativo

- **Threads** esta diseñado para valorar el rendimiento del planificador, mas concretamente, cuando varios hilos compiten por las mismas variables de exclusión mutua, de aquí en adelante *mutex*, creando condiciones de carrera. Para ello, la herramienta genera un número específico de hilos y *mutex* para a continuación hacer que los hilos comiencen las peticiones para bloquear el *mutex*.

Una vez el *mutex* ha sido bloqueado el hilo ocupa la CPU un tiempo, hasta que es puesto de nuevo en la cola de ejecución desbloqueando el *mutex*. Estas operaciones bloquear/ocupar CPU/desbloquear son repetidas varias veces creando un bucle. El test ofrece dos opciones - - *thread-yields* indica el número de veces que se realizará el bucle por cada petición y - - *thread-locks* que indicará el número de *mutex* a crear.

```
sysbench - - num-threads= "número de hilos" - - test=threads - - thread-yields= "veces que se repitira el bucle" - - thread-locks= "número de mutex"
```

- El propósito del test **mutex** es medir la implementación de las variables de exclusión mutua. Para ello, emula una situación en la que todos los hilos corren concurrentemente adquiriendo el mutex por periodos cortos. Este modo dispone de tres opciones que permiten especificar el número de *mutex*, número de bloqueos de *mutex* por cada solicitud y el número de iteraciones a realizar antes de adquirir el bloqueo.

```
sysbench - - num-threads= "número de hilos" - - test=mutex - - mutex-num= "número de mutex" - - mutex-locks= "número de bloqueos" - - mutex-loops= "número de iteraciones"
```

- El modo **memory** se emplea para medir los accesos secuenciales de lectura y escritura en memoria. Dependiendo de las opciones empleadas cada hilo realizará un bloqueo global, o bien local, para todas las operaciones de memoria. Este modo dispone de las siguientes opciones:

```
- - memory-block-size → Tamaño del bloque de memoria a usar.  
- - memory-scope → Con dos valores posible global o local .  
- - memory-total-size → Tamaño de los datos a transferir.  
- - memory-oper → Tipo de operación de memoria con dos valores posibles  
read y write
```

- Para las pruebas de rendimiento de E/S se utiliza el modo **fileio** que produce cargas de trabajo de E/S. En una primera etapa *sysnbench* crea un número específico de archivos con un tamaño determinado y a continuación cada hilo realiza operaciones de E/S sobre esos archivo. Este modo permite realizar comprobaciones de *checksum* con la opción - - *validate*.

Para ello, en cada operación de escritura el archivo se escribe con datos aleatorios, tras lo cual, se calcula su *checksum* y se almacena junto al *offset* de

su bloque. Después, en cada operación de lectura se valida cada bloque con su *offset* almacenado y se compara el *checksum* calculado con el almacenado.

Este modo soporta los siguientes tipos de operaciones: de lectura/escritura: *seqwr*, escritura secuencial; *seqrewr*, re-escritura secuencial; *seqrd*, lectura secuencial; *rndrd*, lectura aleatoria; *rndwr*, escritura aleatoria y *rndrw* lectura/escritura aleatoria.

Si la plataforma sobre la que trabaja lo soporta, también se encuentran disponibles opciones para especificar el modo en que se accederá los archivos: un modo E/S asíncrono *async*, modo *mmap()* lento y modo *mmap()* rápido.

El test *fileio* dispone de varias opciones que permiten afinar este tipo de test, estas opciones se exponen a continuación:

- - *file-num* → Número de archivos que se crearán.
- - *file-block-size* → Tamaño de los bloques de las operaciones de E/S
- - *file-total-size* → Tamaño total de los archivos
- - *file-test-mode* → Tipo de test, *seqwr*, *seqwr*, *seqrd* ... etc
- - *file-io-mode* → Tipo de E/S si lo soporta la plataforma *sync*, *async*, *fastmap* y *slowmap*.
- - *file-async-bcklog* → Número de operaciones asincronas a encolar por cada hilo.
- - *file-extra-flags* → Opciones adicionales para *open(2)*
- - *file-fsync-freq* → Realizar *fsync* después del número de peticiones especificadas, por defecto 100
- - *file-fsync-all* → Hacer *fsync* después de cada operación de escritura
- - *file-fsync-end* → Hacer *fsync* al final del test
- - *file-fsync-mode* → Especifica el método de sincronización empleado, los posibles valores son *fsync* y *fdatasync*
- - *file-merged-request* → Indica el número de peticiones máximo que se mezclarán, el valor por defecto es cero.
- - *file-rw-ratio* → Ratio de E/S para test de E/S combinada aleatorios

Para la ejecución correcta de este tipo de test es necesario ejecutar tres operaciones, un ejemplo de posible ejecución sería el expuesto a continuación:

```
sysbench - - num_threads = "Número de hilos" - - test = fileio - - file-total-size =  
"Tamaño de los archivos" - - file-test-mode = "Modo de ejecución del test" prepare
```

```
sysbench - - num_threads = "Número de hilos" - - test = fileio - - file-total-size =  
"Tamaño de los archivos" - - file-test-mode = "Modo de ejecución del test" run
```

```
sysbench - - num_threads = "Número de hilos" - - test = fileio - - file-total-size =  
"Tamaño de los archivos" - - file-test-mode = "Modo de ejecución del test" cleanup
```

- El último test disponible en *sysbench* es el **oltp**, que está diseñado para evaluar el rendimiento en bases de datos. En una primera fase se crea una tabla que a continuación se rellena con un número especificado de filas que serán accedidas según varios modos en la ejecución subsiguiente.

En el modo simple cada hilo ejecuta una consulta SQL sencilla a la tabla del tipo *SELECT c FRM sbtest WHERE id=N*.

El modo transaccional avanzado realiza transacciones en la base de datos para cada uno de los hilos, si la BD soporta las expresiones *BEGIN/COMMIT* estos se usarán para parar e iniciar las transacciones. En el resto de casos la herramienta empleará las declaraciones *LOCK TABLES/UNLOCK TABLES* . Si se borra alguna fila en una transacción esta se repondrá en la misma transacción, por lo que este test en ningún caso destruye los datos y puede ejecutarse múltiples veces en la misma tabla. Las consultas realizadas por cada una de las transacciones dependerán de las opciones seleccionadas, estas pueden consultarse a continuación.

El modo no-transaccional es muy similar al modo simple pero permite seleccionar la consulta que se realizara, sin embargo, este modo no preserva los datos de la tabla, por lo que se hace necesario ejecutar los comandos *prepare* y *cleanup* entre sucesivas ejecuciones del test. Los diferentes tipos de consulta permitidos pueden ser consultados mas detalladamente en la web del proyecto^[7].

A continuación, se enumeran las diferentes opciones disponibles para este tipo de test, estas se muestran en dos grupos, el primero de ellos comprende las opciones disponibles para todo tipo de drivers, el segundo muestra las opciones específicas para el driver de MySQL.

- *oltp-test-mode* → Los valores posibles son *simple, complex* y *notrx*
- *oltp-read-only* → Modo de solo lectura, no se ejecutarán consultas de actualización borrado o adición.
- *oltp-skip-trx* → Omitir declaraciones de tipo *BEGI/COMMIT*
- *oltp-reconnect-mode* → Modos de reconexión. Los posibles valores son: *session, query, transaction, random*.
- *oltp-range-size* → Rango de tamaños para el rango de consultas
- *oltp-point-selects* → Número de consultas de selección de puntero en una *solo* transacción
- *oltp-simple.ranges* → Número de consultas de rango simple en una sola transacción
- *oltp-sum-ranges* → Número de consultas de rango *SUM* en una simple transacción
- *oltp-order-ranges* → Número de consultas de rango *ORDER* en una simple transacción
- *oltp-distinct-ranges* → Número de consultas de rango *DISTINCT* en una simple transacción
- *oltp-index-updates* → Número de consultas *UPDATE* de índice en una sola transacción.
- *oltp-non-index-updates* → Número de consultas *UPDATE* de no índice en una sola transacción
- *oltp-nontrx-mode* → Tipo de consultas para modos de ejecución no transaccionales. Los posibles valores son: *select, update_key, update_nokey, insert, delete*
- *oltp-connect-delay* → Tiempo de retardo en microsegundos después de cada conexión a la base de datos
- *oltp-user-delay-min* → Tiempo de retardo mínimo en microsegundos *después* de cada solicitud
- *oltp-user-delay-max* → Tiempo máximo de retardo en microsegundos después de cada conexión

- *olpt-table-name* → Nombre de la tabla para pruebas
- *olpt-table-size* → Número de filas en la tabla de pruebas
- *olpt-dist-type* → Distribución de los números aleatorios, existen tres opciones *uniform*, *gauss* y *special*. Con la distribución especial un porcentaje de números es generado en un porcentaje especificado de casos.
- *olpt-dist-pct* → Porcentaje de valores que serán tratados como “especiales”
- *olpt-dist-res* → Porcentaje de casos en el que son generados los valores especiales.
- *db-ps-mode* → Si el driver de la BD soporta el API PS se utilizarán declaraciones preparadas en el lado servidor, en caso contrario se emplearán las preparadas en el lado cliente

Opciones específicas para el driver de MySQL:

- *mysql-host* → Lista de *host* separada por comas. La herramienta distribuirá las conexiones entre los *host* especificados utilizando una lógica *round-robin*¹³. Es importante tener en cuenta que todos los *host* deben compartir tanto el puerto de funcionamiento como la contraseña.
- *mysql-port* → Puerto del servidor de MySQL
- *mysql-socket* → Archivo de *socket* Unix para comunicar con el servidor de MySQL
- *mysql-user* → Usuario de MySQL
- *mysql-password* → Contraseña de MySQL
- *mysql-db* → Nombre de la BD de MySQL
- *mysql-table-engine* → Tipo de tabla de prueba, los posibles valores son *myisam*, *innodb*, *heap*, *ndbcluster*, *bdb*, *maria*, *falcon* y *pbxt*
- *mysql-ssl* → Uso de conexiones SSL
- *myisam-max-rows* → Opción *MAX_ROWS* para las tablas MyISAM
- *mysql-create-options* → Opciones adicionales pasadas a *CREATE_TABLE*

b) b_eff_io

Para el *benchmarking* de E/S la herramienta elegida es *b_eff_io*. Una herramienta que permite obtener el ancho de banda alcanzable de E/S de los datos transferidos de memoria a disco con aplicaciones de procesamiento paralelo. Emplea para ello MPI con diferentes patrones de acceso y diferentes longitudes de *buffer*. Ha sido desarrollada tratando de seguir las reglas sobre *benchmarking* para MPI descritas por Bill Gropp, Ewing Lusk y Rolf Hempel¹⁴

La razón de emplear una herramienta de *benchmarking* diferente se debe a que muchos estudios de rendimiento se hacen para evaluar los límites en las capacidades del *hardware* y del sistema de archivos, centrándose en bajo que circunstancias ha de trabajar un sistema para obtener su máximo rendimiento. Sin embargo, hay situaciones, como la que nos ocupa, en que evaluamos el rendimiento del sistema al ejecutar una aplicación en concreto y las necesidades de esta aplicación primarán

¹³ [Round - Robin](#)

¹⁴ <http://www.mpi-forum.org/docs/mpi-11-html/mpi-report.html>

sobre las capacidades del sistema.

Esta herramienta tiene en cuenta las especificaciones del estándar MPI para operaciones de E/S descritas en la interfaz MPI-I/O, permitiendo obtener mediciones que permitan optimizar el rendimiento de MPI. Esta optimización se consigue ajustando el ancho de banda de E/S de todo el sistema en base a las lecturas proporcionadas por la herramienta.

Un estudio de rendimiento en entornos de procesamiento paralelo debe tener en cuenta una gran cantidad de parámetros, los creadores de `b_eff_io` han identificado y dividido estos parámetros en seis posibles categorías:

- Parámetros de aplicaciones:
 - Tamaño de los fragmentos contiguos en memoria.
 - Tamaño de los fragmentos contiguos en disco, que pueden ser diferentes en los casos en los que los patrones de acceso sean *scatter/gather*
 - Número de fragmentos contiguos a los que se accede en cada llamada a una rutina de lectura/escritura.
 - Tamaño de archivo
 - Esquema de distribución (segmentado o *long strides*, *short strides*, aleatorio o regular, archivos separados para cada nodo)
 - Si el tamaño de los fragmentos y su alineación están correctamente formados.
- Aspectos de uso:
 - Número de procesos en uso.
 - Número de procesadores paralelos e hilos para cada proceso.
- Las interfaces de E/S, en el caso de `b_eff_io`, siempre utilizan soluciones MPI-I/O en detrimento de otras como Posix I/O por considerarla una herramienta portable y de implementación óptima.
- Aspectos ortogonales:
 - Métodos de acceso (primera escritura, escritura y lectura).
 - Método de posicionamiento (puntero individual o compartido).
 - Coordinación (acceso colectivo o no colectivo).
 - Sincronismo (bloqueante o no bloqueante).
 - Si los archivos son abiertos o no como *unique*.
 - Método de consistencia para los conflictos de acceso. Como `b_eff_io` no combina operaciones de computación con operaciones de E/S, solamente se emplean llamadas bloqueantes, por lo tanto, no habrá una gran diferencia entre emplear punteros individuales o compensaciones explícitas.
- Parámetros del sistema de archivos:
 - Nodos por procesador usados como servidor de E/S.
 - Cantidad de memoria usada como espacio de intercambio por cada nodo de la aplicación.

- Tamaño de los bloques en el disco.
- Tamaño de la unidad de *striping*¹⁵.
- Número de dispositivos de *striping* en uso.
- Aspectos adicionales de *benchmarking*:
 - Factores de repetición
 - Métodos para calcular, usando valores promedio, los aspectos anteriores.

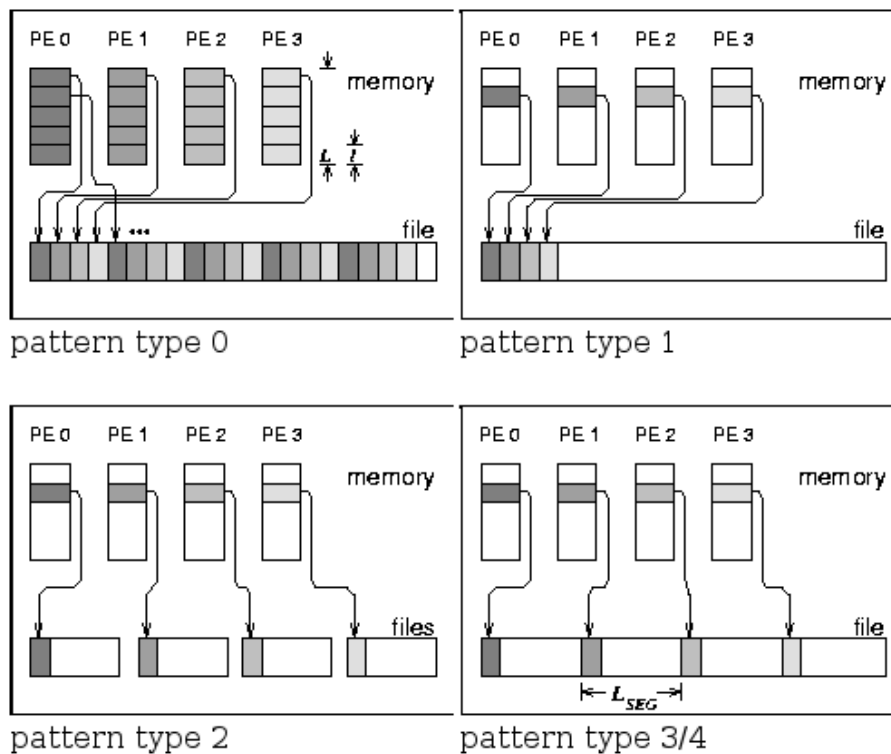
Revisar cada uno de las posibles combinaciones que ofrecen los aspectos anteriores podría resultar en análisis excesivamente largos. Los desarrolladores han considerado que es posible alcanzar un resultado útil en un tiempo de entre diez a quince minutos. Es importante, sin embargo, que no existan aplicaciones en ejecución que usen una parte significativa del ancho de banda. En este caso es posible alcanzar el ancho de banda máximo sin emplear el número total de procesadores.

La medida de unos 10 minutos proviene del empleo MPI-I/O que en este tiempo se espera alcance el cincuenta por ciento del ancho de banda disponible. De esta manera se puede suponer que al cabo de 10 minutos las herramienta habrá transferido al menos la mitad de la memoria disponible en el sistema. Además, para ser capaz de alcanzar unos resultado muy rápidos, la herramienta en lugar de repetir el mismo experimento varias veces emplea una media ponderada sobre un conjunto medio de experimentos repetidos varias veces llamados patrones. Estos patrones recogen diferentes criterios para los parámetros vistos anteriormente en sistemas de procesamiento paralelo. En general los diferentes experimentos se hacen controlando cada uno de los siguientes aspectos:

- Grupo de fragmentos.
- Métodos de acceso (escritura inicial, escritura y lectura)
- Cinco diferentes tipos de patrones:
 - Tipo-0: *Strided access*¹⁶ colectivos, dispersión de grandes fragmentos de la memoria al disco.
 - Tipo-1: *Strided access* colectivos pero una operación de lectura o escritura por cada partición de disco.
 - Tipo-2: Acceso no colectivo a un archivo por cada proceso MPI.
 - Tipo-3: Igual que el método tipo 2 pero con los archivos ensamblados en un único archivo segmentado.
 - Tipo-4: Igual que el tipo 3 pero el acceso al archivo segmentado se realiza con rutinas colectivas.
- El tamaño de las fragmentos contiguos se escoge bien formado.
- Diferentes tamaños para las fragmentos, principalmente 1kB, 32kB, 1MB con un máximo de 2MB y un tamaño de memoria por nodo de ejecución de los procesos MPI de 1 /128.

¹⁵ Bandeado o creación de bandas.

¹⁶ [Strided access](#)



Los patrones utilizados son el resultado de combinar un tipo de patrón, con una medida determinada para los fragmentos y alineamiento del primer byte. El alineamiento se encuentra implícitamente definido por los datos escritos por todos los patrones previos en el mismo tipo de patrón. Cada uno de estos patrones es examinado repitiendo el patrón durante un tiempo determinado.

Para asegurar que todos los procesos paran en la misma iteración, el proceso root envía la señal a todos los nodos tras una sincronización de barrido. El algoritmo de terminación asume en esta operación que la difusión es al menos 10 veces más rápida que cualquier operación de lectura o escritura.

Para los patrones en los que se emplean archivos segmentados, en los que debe existir un tamaño de segmento predefinido, la herramienta calcula ese valor a partir de los resultados obtenidos en los test realizados con los otros dos patrones. El tamaño del segmento se calcula como la suma de los tamaños de los fragmentos multiplicados por un factor de repetición obtenido en los test realizados con esos patrones.

El almacenamiento en cache pueden ser problemático, ya que las mediciones no son capaces de determinar si los datos se han escrito realmente en el disco. Para garantizar que los datos se hayan escrito a disco es posible emplear *MPI_File_sync*. Sin embargo, este comando solo garantiza que otros procesos puedan leer los datos pero no que estos se hayan escrito en un medio permanente. La única manera de asegurar que los datos se han escrito a disco es emplear tamaños mayores que la capacidad de almacenamiento cache del sistema.

Otro punto importante es que la herramienta trata de realizar los test empleando para ello el *flag unique*, de manera que los ficheros solo sean leídos o escritos por ella. No es posible utilizar la opción *MPI_MODE_UNIQUE_OPEN* porque esto permitiría retrasar la ejecución de *MPI_File_sync*. Para solucionarlo la herramienta trata de asegurar que en sistemas en los que la memoria se puede escribir en 10-15 minutos al menos una tercera parte estará escrita por *b_eff_io*

Los resultados de los test para cada partición se muestran como un valor, llamado valor *b_eff_io*. El valor de *b_eff_io* para una partición está definido como la suma de todos los bytes transferidos dividido por el tiempo total de transferencia. El valor total del sistema vendrá dado por el máximo *b_eff_io* de una sola partición.

Para que esta herramienta funcione correctamente es necesario tener instaladas las herramientas de desarrollo de openMPI pues, en primer lugar, ha de ser compilada con la librería de openMPI para el desarrollo de aplicaciones MPI en C. Los comandos relativos a su compilación y prueba serán explicados a continuación:

```
mpicc  
-o b_eff_io [-D without shared] b_eff_io.c -lm
```

La opción *-o* de *mpicc* se emplea para combinar en un solo comando la compilación y el enlace proporcionando un ejecutable. Una vez hemos compilado la herramienta, y disponemos de un ejecutable, ya podremos realizar las pruebas, el comando empleado para ello es:

```
mpirun -np "Número de procesos" ./b_eff_io -MB "Cantidad de MB de memoria por nodo"  
-MT "Cantidad de MB de memoria total del sistema" [Resto de opciones]
```

O también:

```
mpiexec -np "Número de procesos" ./b_eff_io -MB "Cantidad de MB de memoria por nodo"  
-MT "Cantidad de MB de memoria total del sistema" [Resto de opciones]
```

- np* → "Número de procesos"
- MB* → "Cantidad de MB de memoria por nodo"
- MT* → "Cantidad de MB de memoria total del sistema"
- noshared* → "Sustituye a el puntero compartido por un puntero individual en los patrones de tipo 1"
- nounique* → "Elimina *MPI_UNIQUE_OPEN* en cada archivo abierto"
- rewrite* → "Realizar reescritura entre las operaciones de escritura y lectura en cada archivo abierto"
- keep* → "Mantener todos los archivos de benchmarking cerrados hasta el siguiente test"
- N* → "Número de procesos"
- T* → "Tiempo previsto"
- p* → "Ubicación del sistema de ficheros que vamos a comprobar"

- i → “Información del fichero”
- e → “Número de errores impresos en pantalla de cada patrón”
- f → “Prefijo del protocolo de archivo y del protocolo de sumario”

Ambos comandos sirven para comenzar el *benchmarking*. Por último la herramienta proporciona un comando para generar diagramas *beffio_eps*.

c) httperf

Esta es una herramienta pensada para la realización de pruebas de rendimiento en servidores web. No se trata simplemente de una única prueba, si no que esta pensada para poder realizar una gran variedad de pruebas diferentes. Su origen surge de la necesidad de realizar pruebas de rendimiento mas ajustadas a la realidad, pues otras herramientas empleaban un número demasiado pequeño de clientes cuando el crecimiento de Internet implica clientes en cantidades mucho mayores.

Los problemas que tuvieron que hacerse frente en el desarrollo de esta herramienta fueron principalmente tres : en primer lugar los servidores web son sistemas distribuidos, segundo HTTP usa patrones de acceso para los que TCP no está diseñado, y tercero el rápido desarrollo de las tecnologías web que podían dejar obsoleta la herramienta. Por estos tres motivos httperf se pensó con el objetivo de realizar una herramienta escalable y con un buen rendimiento.

El primer obstáculo se consideró cubierto escribiendo la herramienta en lenguaje C y prestando especial atención a los problemas de rendimiento, el segundo, que trata de prever futuras modificaciones de la herramienta, se cubrió dividiéndola en tres partes: el núcleo del motor HTTP, la generación de carga de trabajo y las colecciones estadísticas.

El motor HTTP es el que maneja todas las comunicaciones con el servidor y como tal maneja la conexión, manejo de respuestas, y la generación de solicitudes. El generador de carga es el encargado de realizar las llamadas HTTP apropiadas en los momentos idóneos para generar carga en el servidor. La última de las partes, la colección de estadísticas, es la encargada de la recolección de datos y producir las estadísticas de los resultados del rendimiento. La comunicación entre estas tres partes se realiza a través de un sistema sencillo de señales, de manera que cuando se produce algún evento, se genera una señal que puede ser detectada por un manejador de eventos activado para ello.

Entrando mas en detalle en el generador de carga, este soporta dos modos diferentes, a saber: generación de solicitudes y generación de URL. El primero, a su vez, se subdivide en dos modos diferentes:

Generando conexiones a intervalos regulares y a continuación utilizándolas para realizar un número específico de llamadas.

Emulación del comportamiento típico de la actividad, a través de un navegador. Para lo cual, crea sesiones a intervalos regulares consistentes en un número específico de ráfagas de llamadas.

El generador URL también funciona en dos modos distintos:

Generando la cadena URL con la que se desea acceder al servidor una y otra vez.

Manejando un conjunto de URLs sobre el que moverse, asumiendo una página web organizada en forma de árbol.

Como se ha comentado en el primer párrafo uno de los retos de la herramienta es generar una cantidad de carga suficiente al servidor capaz de sobrecargarlo, y mantenerla en el tiempo. El problema estriba en que una vez que se ha logrado esta carga el cliente seguirá produciendo hasta quedarse sin recursos, por lo tanto, siendo incapaz de generar nuevas solicitudes al servidor. Para solucionar este problema `httperf` fija un tiempo de espera máximo para las solicitudes al servidor, que puede ser especificado en su ejecución, de tal manera que los clientes nunca agoten completamente sus recursos o en el peor de los casos lo hagan solo durante el tiempo de espera especificado.

Es importante tener en cuenta todos los límites que afectan a los clientes a la hora de mantener una carga en el servidor. Los desarrolladores de la herramienta han determinado tres elementos que provocan cuellos de botella en los clientes además de las limitaciones impuestas por sus CPU. Estas limitaciones son: tamaño del puerto TCP, número de descriptores de archivos abiertos y memoria disponible para los *buffer* de los *socket*.

El primer cuello de botella viene determinado por el protocolo TCP, los puertos TCP tienen un tamaño de 16 bits y de entre todos los puertos disponibles hay 1024 reservados para operaciones privilegiadas. Esto implica que la herramienta dispone de 64.512 puertos. Esta puede parecer una gran cantidad, sin embargo, se debe considerar que un puerto TCP no puede ser reutilizado mientras se encuentra en el estado *TIME_WAIT*, lo que puede provocar una disminución en la carga ofrecida por el cliente. El cálculo manejado por los desarrolladores para un tiempo medio de espera de un minuto es de 1075 solicitudes por minuto para cada cliente.

En algunos sistemas se presenta un problema adicional a este respecto, y es que algunas implementaciones de TCP limitan los *sockets* que no están vinculados a una dirección concreta a los puertos efímeros. Estos van desde el 1024 al 5000 lo que reduce enormemente la carga que un cliente es capaz de generar. Para solucionar este escollo `httperf` genera su propio mapa de puertos, sin embargo, esta solución no es óptima aunque en la práctica funciona bien, pues puede resultar en que la herramienta

crea que un puerto se encuentra disponible cuando en realidad no lo esta.

Cuando hablamos del número de descriptores de archivo abiertos, en este caso, nos referimos al número máximo permitido para cada proceso. Los descriptores pueden ser reutilizados casi inmediatamente y su valor oscila, según el SO, entre 256 y 2048, por lo que el cuello de botella no viene producido por el tiempo de espera de TCP si no por el de `httpperf`. Por ejemplo, suponiendo un valor de cinco segundos y un límite de 2000 descriptores de archivo abiertos, se podría mantener un rango de unas 400 solicitudes por segundo. Estos valores pueden ser modificados en el SO para evitar un posible cuello de botella pero se debe tener en cuenta que esto acorta la vida de las conexiones TCP. En el caso de este proyecto el número de descriptores estaba en 1024, y se amplió a 2048 con objeto de las pruebas.

En cuanto a los `socket`, este cuello de botella viene determinado por la cantidad de memoria que los clientes pueden proporcionarles a los `buffer`. La memoria utilizada para los `socket` de cada conexión supone la mayor carga de la herramienta, el tamaño máximo de cada uno de ellos viene fijado y su valor es de cuatro KB los `buffer` de envío, y dieciséis KB los de recepción. En base a estos valores límite, el número de conexiones posibles vendrá determinado por la memoria disponible dividido los veinte KB máximo que supone cada `socket`.

Para terminar, en cuanto a la implementación y las decisiones de diseño de la herramienta, convendría tener en consideración una característica más. Durante su desarrollo se detectó que la herramienta se veía afectada por la granularidad en los procesos, la cual no permitía afinar como sería deseable la generación de carga. A fin de evitar la dependencia del SO en el manejo de la granularidad, `httpperf` ejecuta un bucle cerrado que comprueba la actividad de E/S de la red a través de `select()`, y lleva un registro del tiempo real utilizando `gettimeofday()`. Esto implica que la herramienta consume, en sistemas monoprocesador, todos los ciclos de CPU disponibles.

El planteamiento para medir los rendimientos de la herramienta trata de evitar ofrecer un valor total como resultado de una larga consecución de test. La obtención de un único valor resulta en la posible omisión de datos sobre la evolución y fluctuación del sistema, por lo que los desarrolladores se plantearon obtener muestras cada cinco segundos, de manera que en un periodo no muy prolongado de pruebas se pueda obtener una idea más clara. La herramienta deja en manos del usuario iniciar en cada una de las máquinas las pruebas y realizar luego una media de todas ellas.

A continuación se presenta la descripción de la herramienta y de cada una de las opciones disponibles, así como algunas consideraciones para su ejecución recogidas en su manual¹⁷.

`httpperf`

- - *add-header*= “String” → “Especifica que se incluya el string pasado como una cabecera de solicitud adicional, es necesario especificar el carácter de finalización por ejemplo usando /n”
- - *burst-length*= “longitud de las ráfagas” → Cada ráfaga consiste en N llamadas al servidor
- - *no-host-hdr* → “Indica que la cabecera 'Host' no debe incluirse cuando se realice una petición HTTP”
- - *num-calls*= “Número de llamadas” → En sesiones orientadas a cargas de trabajo se emplea la opción - *wsess*”.
- - *client*= “Cliente” / “Total clientes” → “Indica que la herramienta está ejecutándose en el cliente I de un total de N cliente.”
- - *close-with-reset* → “Opción para cerrar las conexiones TCP mandando RESET en lugar del seguir el three hand shake, esto es útil para sistemas BSD que ralentizan el ritmo de las conexiones”
- - *d| - debug*= “Nivel” → “Fija el nivel de depuración, a mayores valores de N mayor nivel de detalle”
- - *failure-status*= “Valor” → “Especifica un valor N de respuesta HTTP que será tratado como fallo”
- *h | - help* → “Imprime las opciones”
- - *hog* → “Indica que se usarán tantos puertos como sean necesarios, si no se indica se usarán solo los puertos efímeros”
- - *http-version*= “String” → “Especifica la versión que se incluirá en la solicitud enviada al servidor, por defecto la versión es la 1.1”
- - *max—connections*= “Número” → “Como mucho se crearán N conexiones por sesión, esta opción es significativa combinada con - *wsess* y - *wsesslog*”
- - *max-piped-calls*= “Número” → “Máximo N conexiones emitidas antes de obtener una respuesta, esta opción es significativa combinada con - *wsess* y - *wsesslog*”
- - *method*= “String” → “Indica el método utilizado al emitir una petición HTTP, el método por defecto es GET”
- - *nums-conns*= “Número” → “Número total de conexiones a crear”
- - *period*=*[d|u|v]* T1,T2 → “Al igual que - *rate* sirve para proporcionar el intervalo entre conexiones pero con mas flexibilidad. El primer parámetro indica la distribución del tiempo. Si no se especifica el valor por defecto es d, determinista, la opción d significa uniforme y la opción v permite fijar un diferentes intervalos en la forma T1,D1,T2,D2...”
- - *port*= “Puerto” → “Puerto en el que escucha el servidor web, por defecto el 80”
- - *print-reply*=*[header|body]* → “Imprime por la salida estandar el cuerpo, la cabecera o el sumario de la respuesta HTTP”
- - *rate*= “Tiempo” → “Tiempo de intervalo entre las conexiones”
- - *recv-buffer*= “Tamaño” → “Especifica el tamaño máximo del buffer del socket de recepción”
- - *retry-on-failure* → “Opción significativa solo en sesiones de carga, si una llamada especifica devuelve error, se reintenta aunque suponga el fallo de la sesión”
- - *send-buffer*= “Tamaño” → “Especifica el tamaño máximo del buffer del socket de envio”
- - *server*= “String” → “Hostname del servidor HTTP”
- - *server- name*= “String” → “Nombre que aparece que aparece en la cabecera HTTP como host, si no se especifica se utiliza el valor de - - *server*”
- - *session-cookie* → “Si se utiliza se activa la administración de cookies”
- - *ssl* → “Especifica que las comunicaciones deben usar SSL”
- - *ssl-ciphers*= “Lista” → “Lista de cifrados que se pueden utilizar con - *ssl*, por defecto si

no se especifica esta opción se emplea SSLv3”

- *-ssl-no-reuse* → “No reutilizar sesiones ssl abiertas anteriores”

- *-timeout* = “Tiempo” → “Tiempo que espera la herramienta para recibir una respuesta”

- *-think-timeout*= “Tiempo” → “Tiempo máximo que el servidor podrá necesitar para comenzar a contestar. Este tiempo se suma al valor de espera por defecto”

- *-uri*= “String” → “Indica a que URI debe accederse”

-v | - *-verbose* → “Modo verbose”

-V | - *-version* → “Imprime la versión de httpperf por la salida estandar”

- *-wlong*=[y|n], “Archivo” → “Se emplea para generar una secuencia de URI a los que acceder. El archivo pasado como valor tiene que contener una lista de las URI a las que se quiere acceder. Si el primer parámetro tiene el valor 'y' al llegar al final de la lista se comenzará de nuevo, en caso de estar fijado como 'n' al llegar al final de la lista se detiene”

- *-wssess* “Sesiones”, “Llamadas por sesión”, “Tiempo de espera entre sesiones” → “Indica que se deben realizar las mediciones para sesiones, consistentes en secuencias de ráfagas, y no para valores individuales.”

- *-wssesslog* “Sesiones”, “Tiempo de espera entre ráfagas” , “Archivo” → Similar a *-wssess* , en esta opción se acepta como parámetro un archivo que permite indicar tiempos de espera, secuencia de URIs, método y tamaño de las ráfagas.”

A continuación se muestra un ejemplo de como debe ser al archivo usado extraído de la página del manual de httpperf:

```
# session 1 definition (this is a comment)
/foo.html think=2.0
/pict1.gif
/pict2.gif
/foo2.html method=POST contents='Post data'
/pict3.gif
/pict4.gif

# session 2 definition
/foo3.html method=POST contents="Multiline\ndata"
/foo4.html method=HEAD
```

- *-wset* “Número”, “Tiempo de espera entre la generación de URIs” → “Con esta opción se genera una lista de URIs a un ritmo especificado a las que se va accediendo secuencial mente. El formato de las URIs generadas será de la forma: prefijo / Dirección.html done el prefijo es el prefijo de la URI especificada con la opción *-uri* y la dirección es generada automáticamente”

3.3. **Hardware**

Para el presente proyecto, hemos dispuesto de diferentes medios de hardware que han permitido evaluar el rendimiento de la virtualización sobre distintos soportes de almacenamiento, tanto en sus operaciones de E/S, como empleado como SWAP. Concretamente los medios empleados han sido tres: un disco HDD, un disco SSD y un NVRAM de la marca Fusion-io. Con objeto de establecer una base clara sobre las diferencias entre este tipo de soportes a continuación se expone una descripción de cada uno de ellos.

a) HDD

HDD son las siglas de *Hard Disk Drive* conocido comúnmente como disco rígido, y actualmente el formato mas popular sobre todo en el entorno domestico. Este tipo de almacenamiento fue inventado en 1956 por IBM para su computadora IBM 350 y desde entonces a multiplicado su capacidad al tiempo que ha disminuido su coste.

Su funcionamiento consiste en recubrir las superficies de dos platos giratorios con un material magnetizable sobre el que se realiza la grabación empleando un campo magnético de polaridad reversible que magnetiza un cabezal magnético. Inicialmente está información se grababa sobre el disco siguiendo una disposición horizontal pero este método se perfecciono por otros que permiten almacenar la información de manera mas compacta.

Esto fue gracias a dos premios nobel¹⁸, que descubrieron un fenómeno físico conocido como magnetorresistencia gigante, que permitió la construcción de cabezales mas pequeños contribuyendo con ello a que cada bit pudiera ser almacenado en un menor espacio. Fue este descubrimiento en concreto lo que favoreció el crecimiento exponencial en la capacidad de este tipo de discos.

La velocidad en este tipo de almacenamiento está limitada entre otros factores debido a sus componentes mecánicos. Los discos actuales, en sus formatos de 3,5 y 2,5 pulgadas, consisten en una aguja colocada sobre un plato circular sobre el que se encuentra el material magnetizable. Los elementos del disco están actuados por motores eléctricos que permiten la operación.

El tiempo de acceso de estos discos, entendido como el tiempo que tarda el disco en permitir el acceso a los datos, es el que mas relacionado se encuentra con su naturaleza mecánica pues los platos han de girar hasta situar bajo la aguja la zona donde se encuentran los datos a leer o la zona sobre la que vamos a escribir. Su valor, aunque esta definición puede variar, viene dado en función de otros valores de velocidad del disco como la latencia, el tiempo de búsqueda, el tiempo que tarda en situarse el disco y el tiempo de ejecución de los comandos.

18 Albert Fert & Peter Grünberg

Para mejorar el rendimiento de este tipo de discos se han ideado sistemas como el *short-stroking* que consiste en reducir la distancia que la cabeza de la aguja ha de recorrer sobre el plato. Para ello se reduce la capacidad del disco empleando únicamente los anillo exteriores del plato, de esta manera usando solo el exterior la aguja ha de recorrer siempre una distancia pequeña. Este tipo de sistemas suponen que para obtener una mejora en el número de operaciones por segundo, de aquí en adelante IOPS, se deben aumentar el número de discos lo que supone incrementos de espacio y consumo eléctrico.

Este tipo de técnicas puede incrementar el desempeño en IOPS en cinco veces los valores de fábrica del disco. Los valores actuales de estas medidas varían de unos fabricantes a otros pero para ofrecer una idea de ellos se incluyen a continuación los datos proporcionados por Seagate de su producto Constellation.2:

Performance					
Spindle Speed (RPM)	7200	7200	7200	7200	7200
Cache, Multisegmented (MB)	64	64	64	64	64
Interface Access Speed (Gb/s)	6.0, 3.0, 1.5	6.0, 3.0, 1.5	6.0, 3.0, 1.5	6.0, 3.0, 1.5	6.0, 3.0, 1.5
Max. Sustained Transfer Rate (MB/s)	115	115	115	115	115
Seek Time, Average Read/Write (ms)	8.5/9.5	8.5/9.5	8.5/9.5	8.5/9.5	8.5/9.5
Average Latency (ms)	4.16	4.16	4.16	4.16	4.16
Interface Ports	Dual	Dual	Single	Single	Single
RV Tolerance (rad/s ² to 1800Hz)	16	16	16	16	16

Estos discos presentan un problema adicional, el incremento exponencial en la velocidad de los procesadores en los últimos 30 años supone un abismo frente al aumento en la velocidad de este tipo de medios que ha sido marginal. Se hacen pues necesarios medios mas veloces que permitan obtener el máximo beneficio de este aumento en la velocidad de procesamiento.

b) SDD

Los SDD (Solid-state drive) usan circuitos integrados como memoria para el almacenamiento persistente de datos, y no disponen de ningún componente mecánico por lo que su velocidad de acceso es mayor que en los HDD. Para el almacenamiento persistente, este tipo de medio emplea memorias tipo NAND que son capaces de mantener los datos sin necesidad de alimentación eléctrica. Su gran desventaja viene dada por su coste, que es aún hasta diez veces mayor que el de un HDD clásico.

Los componentes principales de un SSD son su controlador y las memorias sobre las que se almacenan los datos. Inicialmente era común usar en este tipo de almacenamiento memorias DRAM, pero en los últimos años es mucho mas frecuente el uso de memoria no volátiles NAND. El controlador de los SSD es un procesador embebido en el disco que ejecuta instrucciones a nivel de código máquina para manejar diferentes funcionalidades como: mapeado de sectores defectuosos, equilibrado del uso de disco, depuración y manejo de las alteraciones en la lectura,

cacheo de lectura y escritura, encriptación y liberación de memoria¹⁹.

Su gran velocidad de acceso permite a los discos SSD disponer de un elevado número de operaciones de lectura escritura por segundo, frente a los discos HDD. Podemos decir, por lo tanto, que aunque su coste es mayor la relación rendimiento coste es mejor, y mas si tenemos en cuenta que las mejoras en las densidades de las memorias y la reducción en los precios las ha hecho mas asequibles en los últimos años .

El ahorro en consumo viene dado principalmente por dos motivos: este tipo de discos como ya hemos comentado no precisa de partes móviles, y además, dado que su rendimiento es mejor, permiten alcanzar el mismo desempeño operacional en IOPS con un menor número de discos. Este punto es aun mas valorable si tenemos en cuenta que los sistemas de mejoras en el desempeño de los HDD suponen una merma de su capacidad, cosa que no sucede con los SDD que con una mejor rendimiento conservan intacta su capacidad.

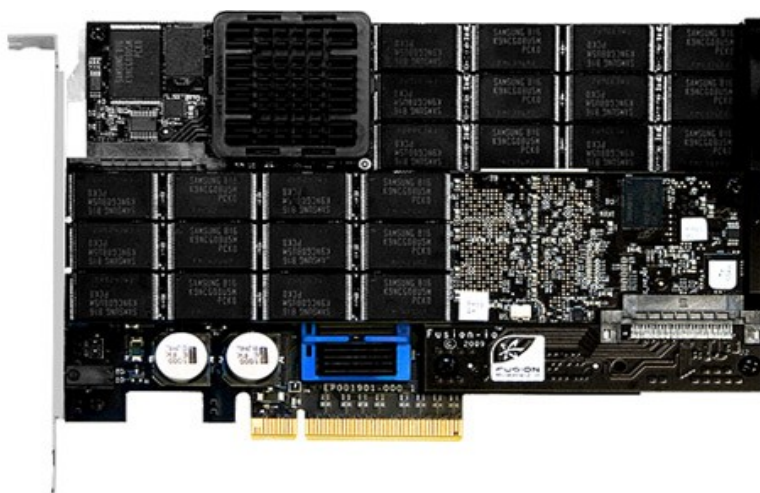
Una desventaja frente a los HDD es la capacidad de almacenaje, en este caso la relación capacidad de almacenaje precio es mucho mas alta, máxime si se trata de datos con un bajo índice de accesos que compense un menor rendimiento de disco, que puede además ser compensado añadiendo mas servidores con mas CPUs o nodos. Otra gran desventaja es la resistencia al fallo, mientras que en un disco HDD que haya fallado hay una gran probabilidad de éxito a la hora de recuperar los datos en los discos sólidos un fallo suele ser catastrófico e implica una perdida total.

La función del controlador es vital para mantener la integridad de los datos, ya que mientras que en un HDD los datos son almacenados linealmente, un disco sólido reubica constantemente los datos para aprovechar al máximo la capacidad. La correcta ejecución de esta y otras funciones desempeñadas por el controlador depende directamente del firmware utilizado, un error en este software podría resultar fatal.

19 Garbage collection

c) Fusion-io ioDrive duo 640Gb

Para el proyecto vamos a contar con un novedoso sistema de hardware fabricado por la empresa Fusion-io con sede en Salt Lake City. El producto en cuestión es un ioDrive duo de 640Gb que ofrece una nueva aproximación al uso de memorias NAND como alternativa a los HDD. Es habitual que los discos SSD empleen el mismo interfaz que un HDD para su conexión, este planteamiento sacrifica algunos de los beneficios de SSD en cuanto a velocidad y latencia al situarlos tras protocolos de almacenamiento pensados para HDD.



Para solventar esto se han planteado otro tipo de interfaces, que permitan alcanzar un nivel superior de rendimiento en el uso de SSD. Uno de estos consiste en situar las memorias NAND sobre un circuito integrado, que se pueda conectar a un puerto PCIe en lugar de en el espacio reservado para los discos, en este caso, el controlador es el RAID embebido. Si bien esta aproximación elimina el cableado y libera algunas capas de complejidad a la primera opción planteada, aun utiliza protocolos de almacenamiento propios de discos HDD como el SAS o SATA, se podría decir, por lo tanto, que la única ventaja obtenida es la liberación de las espacios para discos y la eliminación de cableado.

El ioDrive para evitar este tipo de protocolos de almacenaje proporciona acceso a los datos mediante DMA, liberando así al hardware de mediadores como RAID que fueron pensados para medios mas lentos con agujas y platos que necesitan rotar. Pero no es lo único que esta pensado para este tipo de medios, los procesadores embebidos están pensados para evitar a la CPU tener que esperar por las lentas operaciones de E/S, sin embargo las memorias NAND son lo suficientemente rápidas para que esto no sea necesario, al contrario, este planteamiento supone un cuello de botella al situar su velocidad tras el procesador embebido.

De esta manera se recrea en cierto modo el abismo entre las velocidades de procesamiento y las velocidades de acceso de los discos. Para evitar esto el *ioDrive* simplemente utiliza la CPU, igual que hace el sistema de memoria virtual, para situar tablas de metadatos a las que se puede acceder de manera independiente. Para ello los fabricantes han desarrollado un subsistema operativo virtual llamado VSL (*Virtual Storage Layer*) optimizado para flash . Este SO permite al hardware interactuar directamente con cada nodo de la CPU en paralelo ,como si se tratara de un nuevo nivel de memoria, al mismo tiempo que la presenta como un sistema de almacenamiento basado en bloques. Tratar las memorias NAND como un nuevo nivel de memoria permite tener solamente dos cambios de contexto, independientemente de la cantidad de módulos que se empleen, motivo por el que, en palabras del fabricante, este hardware es capaz de aumentar el rendimiento de forma lineal.

Gracias a todo lo expuesto anteriormente el rendimiento es muy alto, según las especificaciones facilitadas por el fabricante en la web para el modelo que nos ocupa el *ioDrive* es capaz de realizar 93.000 IOPS de lectura, 145.000 IOPS de escritura y 74.000 IOPS combinadas de lectura y escritura. En cuanto a la velocidad de acceso si la comparamos con los datos proporcionados para el HDD de *Seagate* que hemos empleado como ejemplo en el apartado anterior, vemos que la diferencia es abismal, situándose en 4'16 m/s frente a los $30\mu\text{s} = 0'03 \text{ m/s}$ de la *ioDrive*.

Las ventajas de este producto no finalizan con su excelente rendimiento, el fabricante promete además un consumo energético realmente bajo comparado con los HDD. Según datos ofrecidos en uno de sus *white papers* ^[18] el rendimiento del *ioDrive* equivaldría al de 600 HDD paralelos en un SAN, pero requiriendo solo el consumo eléctrico de una única unidad. Esto supondría por lo tanto, una disminución de hasta 10.000 W de consumo producida gracias a una mejor utilización de la CPU, que permite obtener rendimientos equivalentes con una reducción de hasta un 90% en sus consumos.

d) Potenciómetro

Una de los objetivos del presente trabajo consiste en evaluar el consumo que las diferentes tecnologías presentadas suponen unas frente a otras, para ello se ha dispuesto de hardware que permite la evaluación de estos consumos durante la ejecución de las diferentes pruebas.

Concretamente se ha empleado un potenciómetro de la compañía “Watts up”²⁰, el Watts ups Pro. Pueden encontrarse sus especificaciones en la página web de los fabricantes (ver pie de página) pero cabe destacar que proporciona una precisión de +/- 1.5%.

²⁰ <https://www.wattsupmeters.com/secure/products.php?pn=0&wai=512&spec=3>

4. Preparación del entorno y ejecución de las pruebas

4.1. Instalación de KVM

Para instalar KVM en primer lugar necesitamos que el hardware que vamos a utilizar disponga de las herramientas de virtualización. En nuestro caso el equipo si disponía de estas herramientas, pero aunque no fuera así podríamos virtualizar con un rendimiento ostensiblemente menor.

En cualquier caso se puede comprobar si un equipo dispone de estas funcionalidades, que previamente deben haber sido habilitadas en la BIOS. El SO que vamos a emplear es una distribución Linux, concretamente una Fedora 17 y el comando a ejecutar es el siguiente:

```
~$ egrep -c '(vmx|svm)' /proc/cpuinfo
4
```

Si el resultado mostrado por consola es igual o mayor que uno entonces disponemos de herramientas para la virtualización. En este caso podemos ver el valor que se obtuvo fue cuatro y por lo tanto pudimos continuar con la instalación. Otra opción sería el comando que se muestra a continuación, en este caso si simplemente obtenemos una salida sabremos que disponemos de herramientas para la virtualización:

```
~$ egrep '(vmx|svm)' --color=always /proc/cpuinfo
```

```
flags: fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush dts
acpi mmx fxsr sse sse2 ss ht tm pbe syscall nx lm constant_tsc arch_perfmon pebs bts
rep_good nopl aperfmperf pni dtes64 monitor ds_cpl vmx est tm2 ssse3 cx16 xtpr pdcm
lahf_lm dtherm tpr_shadow vnmi flexpriority
```

```
flags: fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush dts
acpi mmx fxsr sse sse2 ss ht tm pbe syscall nx lm constant_tsc arch_perfmon pebs bts
rep_good nopl aperfmperf pni dtes64 monitor ds_cpl vmx est tm2 ssse3 cx16 xtpr pdcm
lahf_lm dtherm tpr_shadow vnmi flexpriority
```

```
flags: fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush dts
acpi mmx fxsr sse sse2 ss ht tm pbe syscall nx lm constant_tsc arch_perfmon pebs bts
rep_good nopl aperfmperf pni dtes64 monitor ds_cpl vmx est tm2 ssse3 cx16 xtpr pdcm
lahf_lm dtherm tpr_shadow vnmi flexpriority
```

```
flags: fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush dts
acpi mmx fxsr sse sse2 ss ht tm pbe syscall nx lm constant_tsc arch_perfmon pebs bts
rep_good nopl aperfmperf pni dtes64 monitor ds_cpl vmx est tm2 ssse3 cx16 xtpr pdcm
lahf_lm dtherm tpr_shadow vnmi flexpriority
```

A continuación procedimos a instalar los paquetes necesarios. Para tener KVM instalado y

funcionando, ejecutamos:

```
~$ sudo yum install kvm libvirt python-virtinst qemu-kvm
```

La salida del comando se ha omitido por ser excesivamente larga (se instalaron hasta 35 paquetes tras resolver dependencias) y no aportar ningún dato adicional. A continuación creamos los enlaces de inicio para el demonio de libvirtd e iniciamos el servicio:

```
~$ sudo systemctl enable libvirtd.service
~$ sudo systemctl start libvirtd.service
```

Comprobamos que todo se encontraba instalado y funcionando ejecutando virsh, para que liste las MV en funcionamiento, si responde aunque sea en blanco; ya que no habia MV instaladas, sabremos que está funcionando:

```
~$ sudo virsh -c qemu:///system list
```

<i>Id</i>	<i>Nombre</i>	<i>Estado</i>

Seguidamente fue necesario deshabilitar SELinux para no tener problemas con los permisos al ejecutar KVM, para ello modificamos el archivo de configuración de SELinux y reiniciamos el sistema para que los cambios tuvieran efecto como se muestra a continuación:

```
~$ sudo vi /etc/selinux/config
```

Una vez abierto el editor buscamos SELinux=enabled y lo modificamos por SELinux=disabled, para a continuación reiniciar el sistema con reboot. Agregamos el usuario al grupo kvm;

```
~$ sudo usermod -G kvm -a raul
```

Después cargamos los módulos de kvm en el Kernel, es importante hacerlo a continuación pues al cargarse lo harán con los permisos ya especificados y si no hemos agregado el usuario al grupo habría que recargar el módulo para que el cambio tuviese efecto:

```
~$ modprobe kvm
~$ modprobe kvm-intel
~$ lsmod | grep kvm
   kvm_intel      132763      0
   kvm            419459      1   kvm_intel
```

Empleando wget obtuvimos una imagen de disco mínima para instalar en las MV guest:


```
~$ wget -c http://cdimage.debian.org/debian-cd/6.0.6/i386/iso-cd/debian-6.0.6-i386-netinst.iso
```

Una vez descargado comenzamos la instalación de la primera máquina virtual utilizando virt-install del paquete virtinst:

```
~$ sudo qemu-img create -f qcow2 /var/lib/libvirt/images/vm00.img 10G

~$ sudo virt-install
      -n vm00
      -r 1000
      --vcpus=1
      --disk path=/var/lib/libvirt/images/vm00.img,size=10
      -l http://ftp.de.debian.org/debian/dists/squeeze/main/installer-i386/
      -w bridge=virbr0
      --nographics
      --noautoconsole
      -x console=ttyS0,115200
```

Dada la naturaleza de la conexión, a través de ssh, fue necesario ejecutar todo en modo consola. Para ello se han añadidos las opciones mostradas mas arriba cuya combinación resultó exitosa. El hecho de emplear la opción - noautoconsole hizo que posteriormente a la ejecución de virt-install tuvieramos que conectarnos a la consola de la MV. Para ello empleamos virsh y por lo tanto ejecutamos:

```
~$ sudo virsh

Welcome to virsh, the virtualization interactive terminal.
Type: 'help' for help with commands
      'quit' to quit

virsh #
```

Una vez aparecio el prompt de virsh ejecutamos console “máquina virtual” y tras pulsar enter nos conectamos a la máquina virtual donde pudimos proceder a la instalación del SO en modo consola. El SO elegido ha sido Debian squeeze. Los pasos para la instalación del SO elegido se omiten por aportar poco al conjunto del trabajo. El disco se ha particionado como ext3 asignado 10'2 GB para el sistema y 488'6 MB como swap.

4.2. Preparación de las máquinas virtuales:

Para poder realizar las diferentes pruebas sin que se encuentren trabajando servicios que no son los evaluados se ha creado una MV para cada uno de estos servicios. Estas MV se utilizaron como semilla a partir de las cuales clonamos el resto de máquinas. A continuación se exponen las configuraciones particulares que se realizaron en cada una de ellas:

a) Servidor HTTP

En este caso al instalar el SO indicamos que la máquina se utilizara como servidor web, por lo que ya se encontraban instalados tanto apache2 como apache2-mpm-prefork. Lo que si fue necesario instalar es la herramienta de benchmarking, httpf. Para la ejecución de las pruebas en cada uno de los clones emplearemos un script²¹. Este script necesita para su ejecución de las direcciones IP de cada clon, las cuales, al ser asignadas por dhcp es posible que cambien de un clonado al siguiente. Por ello instalamos en el host la aplicación *arpscan*. Su ejecución nos devuelve como resultado las IP de todas las máquinas conectadas a la interfaz br0.

b) Servidor de archivos

La instalación fue similar pero en este caso seleccionamos durante la instalación que se incluyera el software necesario para un servidor de archivo. Además vamos a estudiar el rendimiento de MPI, concretamente de MPICH2 versión 1.2 y por lo tanto era necesario que instaláramos todos los paquetes necesarios. Para comenzar, es importante aclarar que, en este caso, se comenzó utilizando dos MV “semilla”: vmMPIMaster y vmMPINode y se creó en ambas un mismo usuario llamado mpiuser. La necesidad de emplear dos diferentes MV vino motivada por la configuración que empleamos.

Para las pruebas relacionadas con el benchmarking de sistemas de archivos utilizamos una topología en estrella, de tal manera que se hacían necesarias configuraciones diferentes para el maestro y para sus nodos. La ejecución de instrucciones entre maestro y nodos inevitablemente se entrecruza, por lo que indicaremos frente a la instrucción si esta se ejecutó en el maestro o en el nodo.

Primero de todo instalamos las herramientas de compilación para lenguaje C en GNU/Debian, estos fueron necesarios para la posterior instalación de MPICH2:

```
Maestro & Nodo → # apt-get install build-essential
```

En vmMPIMaster, de aquí en adelante nodo maestro, instalamos un servidor NFS, para compartir un sistema de archivos donde trabajarán todas las MV, se instalaron por lo tanto los paquetes necesarios:

²¹ Ver descripción mas adelante

Raúl López Martín

```
Maestro → # apt-get install nfs-kernel-server
```

A continuación creamos el sistema de archivos a compartir e incluimos este en el archivo de configuración de NFS /etc/exports:

```
Maestro → # mkdir /mirror  
Maestro → # echo "/mirror *(rw,sync)" >> /etc/exports
```

Para el funcionamiento de MPI era necesario que los nodos se comunicaran entre ellos, esto se hizo a través de ssh y por lo tanto fue necesario instalar los paquetes para que ssh funcionara también, por lo tanto:

```
Maestro & Nodo → # apt-get install openssh-client openssh-server
```

Sin embargo, no solo era necesario que las MV puedan comunicarse por ssh, si no que además debían hacerlo sin emplear contraseña, generamos entonces una clave pública para ello:

```
Maestro → $ ssh-keygen -t dsa
```

Y la incluimos entre los hosts autorizados:

```
Maestro → $ cd .ssh  
Maestro → /.ssh$ cat id_dsa.pub >> authorized_keys
```

Esta clave se exportó a cada uno de los nodos/MV que se utilizaron:

```
Maestro → cat id_dsa.pub | ssh mpiuser@vmOMPINode1 'cat >>  
.ssh/authorized_keys'
```

Una vez configurados NFS y SSH nos restó instalar MPICH, para ello en Debian, tanto en el maestro como en el nodo, ejecutamos:

```
Maestro & Nodo → → # apt-get install mpich2
```

Para la correcta ejecución de el cluster que preparamos configuramos direcciones ip estáticas en cada nodo y en el maestro. Editamos por lo tanto en el maestro como en el nodo /etc/hosts:

```
Maestro & Nodo → # nano /etc/hosts
```

```
127.0.0.1 localhost  
192.168.122.64 vmMPIMaster
```

Raúl López Martín

```
192.168.122.63 vmMPINode0
192.168.122.168 vmMPINode1
192.168.122.56 vmMPINode2
192.168.122.232 vmMPINode3
192.168.122.33 vmMPINode4
```

De momento los comandos y configuraciones que se Debían implementar en el nodo solo habían ejecutado una vez en la MV vmMPINode que clonamos posteriormente. La configuración en /etc/network/interfaces debía ser aplicada en cada una de las MV que se emplearon y por lo tanto en este caso, empleándose tres MV debíamos editar en cada MV:

```
Una vez por MV → nano /etc/network/interfaces
```

```
iface eth0 inet static
address 192.168.122.XX
netmask 255.255.255.0
gateway 192.168.122.1
```

A continuación definimos en el nodo maestro el path de mpich2 para ssh:

```
echo /mirror/mpich2/bin >> /etc/environment
```

Mpich2 precisaba que se encontrara compartido una archivo que liste todos los nodos que componen el cluster. Para ello creamos el archivo mpd.hosts en la compartición NFS:

```
$ touch /mirror/mpd.hosts
```

```
$ nano /mirror/mpd.hosts
```

```
vmMPIMaster
vmMPINode0
vmMPINode1
vmMPINode2
vmMPINode3
vmMPINode4
```

También se precisaba que en el home del usuario que ejecutará mpich2 hubiera un archivo de configuración, solo accesible para el usuario, con una palabra secreta para la ejecución en cada nodo. Este archivo se ha de llamar .mpd.conf y se creo y exporto al nodo empleado como semilla:

```
Maestro → $ echo secretword="palabra_secreta" >> ~/.mpd.conf
```

```
Maestro → $ chmod 600 ~/.mpd.conf
```

Raúl López Martín

```
Maestro → $ echo secretword="palabra_secreta" | ssh mpiuser@vmMPINode4  
'echo secretword=fusionio123 >> ~/.mpd.conf
```

Llegados a este punto pasamos ya a configurar el nodo semilla. En primer lugar editamos `/etc/fstab` para que se montase la partición NFS:

```
Nodo → $ nano /etc/fstab
```

```
vmMPIMaster:/home/mpiuser /home/mpiuser nfs
```

Finalmente nos resta clonar el nodo semilla tantas veces como nodos queríamos utilizar. En este caso el nodo se clono tres veces empleando un script que se detalla mas adelante. Una vez clonados los nodos y configuradas sus direcciones estáticas fue vital modificar el archivo `/etc/hostname` en cada uno de los nodos, dado que de lo contrario MPICH2 no funciona correctamente. Nuevamente empleamos el editor Nano:

```
En cada nodo → $ nano /etc/hostname
```

Una vez hecho esto la compartición NFS dejo de funcionar. Para solucionarlo, según información obtenida del foro de Debian^[24], en cada nodo clonado se ejecuto:

```
$ /etc/network/if-up.d/mountnfs
```

```
$ rm -R /var/run/network/mountnfs
```

Esto resolvió el problema y nos permitió probar la ejecución de MPICH, de acuerdo a la guía de instalación^[25] ejecutamos varias pruebas, en concreto :

```
$ mpd &  
$ mdpboot -n 6  
$ mpdtrace  
$ mpiexec -n 6 /bin/hostname
```

Todas ellas con éxito, también como comprobación adicional creamos un pequeño script `hello_world` a partir del código que fácilmente se puede obtener en la web²² y se ejecuto empleando MPI. También en este caso la ejecución fue exitosa.

c) Servidor de BD

Finalmente no restaba instalar un servidor de base de datos. El procedimiento a seguir es básicamente el mismo que en los casos anteriores, instalamos el SO y seleccionamos la opción para que se instaló un servidor de base de datos. Sin embargo

²² <http://www.mpitutorial.com/mpi-hello-world/>

la base de datos que se instala con Debian squeeze es postgresQL 8.4, y por compatibilidad con sysbench instalamos mysql, para ello:

```
$ apt-get install mysql-server
```

Una vez instalada la base de datos accedimos a ella como root para realizar algunas configuraciones:

```
mysql -u root -p
```

Era necesario que crear el usuario con el que realizamos las pruebas desde el host, por ello una vez en mysql ejecutamos:

```
GRANT ALL PRIVILEGES ON *.* TO 'sqluser'@'localhost' IDENTIFIED BY  
'fusionio123' WITH GRANT OPTION;  
GRANT ALL PRIVILEGES ON *.* TO 'sqluser'@'%' IDENTIFIED BY 'fusionio123'  
WITH GRANT OPTION;
```

Creando de esta manera el usuario sqluser con el que realizamos las pruebas. Finalmente debíamos permitir que se accediera al servidor de BD desde la red, para ello editamos `/etc/mysql/my.cnf` y reemplazamos `bin-address=127.0.00` por `bind-address=0.0.0.0`:

```
# nano /etc/mysql/my.cnf
```

Reiniciamos la base de datos:

```
# /etc/init.d/mysql restart
```

4.3. **Instalación de Iodrive**

La realización de las pruebas con el iodrive duo de 640 GB ha sido necesario proceder a su instalación en el equipo empleado. La instalación tal y como la detallamos aquí no resulto exitosa, los inconvenientes surgidos se detallan mas adelante y en esta sección supondremos un proceso en condiciones óptimas.

Para la instalación del iodrive, es necesario en primer lugar establecer cual es el kernel con el que estamos trabajando, lo obtenemos ejecutando la instrucción de Linux `uname -r`. En el caso de este proyecto no fue necesario pues se nos facilito previamente el driver con el que debíamos trabajar, estos drivers en cualquier caso se encuentran disponibles en la página de soporte de Fusion-io²³.

Puesto que para la versión del software que empleábamos no se disponía de un paquete

²³ <http://support.fusionio.com>

rpm, el archivo del driver consistía en las fuentes de este y con ellas debimos generar el paquete rpm para instalarlo en nuestro sistema. Necesitábamos por lo tanto instalar algunos paquetes previamente antes de pasar a construir el driver, de acuerdo a la guía de instalación procedimos a instalar los prerequisites indicados:

```
$ yum install kernel-devel kernel-headers rpm-build gcc rsync
```

Una vez hecho esto ya podemos pasar a construir el paquete rpm. Puesto que este va a ser instalado en la máquina sobre la que lo construimos, ejecutamos:

```
$ rpmbuild --rebuild iomemory-vsl-<VSL-version>.src.rpm
```

Si se tratará de generar un paquete para otra máquina el comando debería variar. El resultado fue un paquete rpm situado en el mismo directorio en el que estábamos trabajando y que procedimos a instalar:

```
$ rpm -Uvh iomemory-vsl-<kernel-version>-<VSL-version>.x86_64.rpm
```

Tras los cual instalamos el resto de software de soporte proporcionado con las fuentes del driver:

```
$ rpm -Uvh lib*.rpm  
$ rpm -Uvh fio*.rpm
```

Con el driver y software adicional ya instalado podíamos proceder a cargar el driver en el kernel, para ello empleamos el comando *modprobe*:

```
$ modprobe iomemory-vsl
```

Una vez cargado el driver ejecutamos el comando *fio-status* y comprobamos que la unidad de iodrives se encontraba funcionando. El siguiente paso habría sido ejecutar *fio-attach*, sin embargo en este caso, tras un reinicio de la máquina motivado por una actualización del firmware, la ejecución se realizó automáticamente y no fue necesario. Por último montamos la unidad iodrives como un sistema de ficheros, ejecutando:

```
$ sudo /sbin/mdadm --stop /dev/md127
```

```
$ sudo /sbin/mkfs -t ext4 /dev/fioa1
```

```
$ sudo mount /dev/fioa1 /media/iodrives
```

Una vez hecho esto ya teníamos la unidad iodrives montada y disponible para comenzar a realizar las pruebas.

4.4. **Scripts:**

En la preparación de las pruebas hemos podido detectar que algunas de las tareas a realizar son repetitivas. A fin de ganar en tiempo, desarrollamos una serie de script que nos ayudasen con esta clase de tareas. Estos scripts se describen a continuación y se incluye su código con fines aclaratorios.

a) Script para las máquinas virtuales

Para la gestión de las máquinas se desarrolló un script que permitía agilizar la gestión en la creación y destrucción de las MV así como en su inicio y parada. Se incluye el código de este script a continuación para clarificar su funcionamiento de aquí en adelante.

```
#!/bin/bash

#The usage method will be express as firts parameter
#second parameter must be disk location if needed. With start & shutdown options this
parameter must be the VM name
#third is the original VM name. Default location for originals are /var/lib/libvirt/images/

#Number of VM to wotk with
VIRTUAL_MACH=10
#Default location for the original VM
loc=/var/lib/libvirt/images/
#Var
c=0

case $1 in
    #Clone the VM VIRTUAL_MACH times
    clone)
        args=3
        #Check the number of arguments for this option
        if [ $# -ne $args ]
        then
            echo "Three argumentes are expected"
        else
            #If we have all the arguments begin cloning
            while [ $c -ne $VIRTUAL_MACH ]
            do
                #Clone the vm with a random mac
                virt-clone --original $3 --name $3$c --file $2$3$c".img" --mac
                'DE:AD:BE:EF:'$[ ( $RANDOM % 100 ) ]:'$[ ( $RANDOM % 100 ) ]
                c=`expr $c + 1`
            done
        fi;;

```

Raúl López Martín

```
#Destroy and undefine all the VM from 0 to VIRTUAL_MACH and remove their images
destroy)
    args=3
    #Check the number of arguments for this option
    if [ $# -ne $args ]
    then
        echo "Three argumentes are expected"
    else
        while [ $c -ne $VIRTUAL_MACH ]
        do
            virsh destroy $3$c
            virsh undefine $3$c
            rm $2$3$c".img"
            c=`expr $c + 1`
        done
    fi;;
#Start all the VM
start)
    args=2
    if [ $# -ne $args ]
    then
        echo "VM name expected"
    else
        while [ $c -ne $VIRTUAL_MACH ]
        do
            virsh start $2$c
            c=`expr $c + 1`
        done
    fi;;
#Shutdown all the VM
shutdown)
    args=2
    if [ $# -ne $args ]
    then
        echo "VM name expected"
    else
        while [ $c -ne $VIRTUAL_MACH ]
        do
            virsh shutdown $2$c
            c=`expr $c + 1`
        done
    fi;;
*)
    echo "Must specify an option (clone or destroy)";;
esac
```


b) Script para benchmarking de HTTP

Para facilitar el lanzamiento de las pruebas de benchmarking en las diferentes máquinas virtuales, escribimos un script que lanza cada una de las pruebas y dirige los resultados a diferentes ficheros. Para su funcionamiento se ha de pasar al script los últimos dígitos de la dirección IP de cada máquina, de tal manera que se lancen las pruebas en cada una ellas. A continuación se incluye el código de este script:

```
#!/bin/bash

    echo "Benchmark r going to be launch!!!!"

args=$#

while [ "$*" != "" ]
do

    httpperf --hog --server=192.168.122.$1 --port=80 --wssess=4,15,10 > /home/raul/results/IP$1
&
    shift

done

    echo "Benchmarks launched!!!!" &
```

c) Script potenciómetro

Para el uso del potenciómetro se ha dispuesto de un script²⁴ que permitía su control con los comandos:

```
start | catlog | catmeasures | totalenergy | totaltime | stop.
```

Este script ha sido el que ha permitido controlar el potenciómetro con comodidad y obtener posteriormente los valores medidos. No se incluye su código por no haberse desarrollado en el transcurso del presente trabajo.

²⁴ Script escrito por M. Gamell

4.5. Instalando las herramientas de Benchmarking

a) sysbench

La aplicación de benchmarking sysbench ya se encuentra empaquetada para Debian, por lo que se instaló simplemente ejecutando en la MV semilla:

```
$ sudo apt-get install sysbench
```

b) b_eff_io

Para instalar b_eff_io en primer lugar la descargamos de la web de los desarrolladores, como esta web emplea certificado utilizamos la opción `-no-check-certificate` para descargarla usando `wget`:

```
# wget - -no-check-certificate  
https://fs.hlrs.de/projects/par/mpi/b_eff_io/b_eff_io_v2.1.tar.gz
```

Una vez descargado procedimos a seguir las indicaciones que se ofrecen en la web. Descomprimos el contenido del tar-gz

```
# gunzip -c b_eff_io_v2.1.tar.gz | tar -xvf -  
b_eff_io/README  
b_eff_io/b_eff_io.c  
b_eff_io/b_eff_io_eps  
b_eff_io/b_eff_io.man.txt  
b_eff_io/b_eff_io.1  
b_eff_io/man/man1/b_eff_io.1  
b_eff_io/b_eff_io_eps.gnuplot  
b_eff_io/b_eff_io_eps_on1page.dvi  
b_eff_io/b_eff_io_eps_on1page.tex
```

Compilamos la herramienta utilizando las utilidades ofrecidas por openMPI, como ya explicamos en el apartado relativo a b_eff_io:

```
# mpicc -o b_eff_io -D WITHOUT_SHARED b_eff_io.c -lm
```

Y ejecutamos una pequeña prueba para comprobar que funcionaba:

```
# mpirun -np 2 ./b_eff_io -MB 256 -MT 512 -T 10 -p /mirror/scratch
```

```
b_eff_io = 292.751 MB/s on 2 processes with 256 MByte/PE, scheduled time=0.5  
Min, on Linux vmMPI 2.6.32-5-686 #1 SMP Sun Sep 23 09:49:36 UTC 2012 i686,  
NOT VALID (see above)
```

c) httperf

Su instalación fue sencilla ya que también se encuentra como paquete en los repositorios y solo ha sido necesario ejecutar :

```
$ sudo yum install httperf.x86_64
```

4.6. Ejecución de las pruebas

a) httperf

Para las pruebas comenzaremos por httperf, en conjunto ejecutamos 4 pruebas de rendimiento y consumo de los servidores web sobre los diferentes medios. Primero comprobamos el disco HDD con swap en este mismo disco, a continuación repetimos las pruebas activando swap en el disco SSD. Después realizamos las pruebas con las MV en el SSD y swap en el mismo disco, y finalmente en la unidad iordrive con swap en el disco SSD. La prueba se ha definido para tener una duración de aproximadamente unos diez minutos, el comando elegido es el que se expone en el script que hemos creado para estas y se muestra a continuación:

```
httperf --hog --server=192.168.122.XX --port=80 --wsess=4,15,10
```

En la elección de los parámetros se ha tenido en cuenta además, que la prueba fuera lo mas parecida posible al funcionamiento real de un servidor web. Con este objetivo en mente hemos seleccionamos la opción - -wsess, de tal manera que se realizarón 4 sesiones con 15 peticiones cada una a una a frecuencia de 10 segundos entre ellas.

- HDD + HDD swap

En primer lugar clonamos la MV imagen del servidor http que habíamos instalado, el clonado lo realizamos ejecutando el script que hemos descrito anteriormente y el resultado lo almacenaremos en el disco HDD, por lo tanto ejecutamos:

```
$ sudo /home/raul/scripts/./tfc.sh clone /media/hdd/httpImages/ vmHttp
```

Seguidamente iniciamos todas las MV también con el script:

```
$ sudo /home/raul/scripts/./tfc.sh start vmHttp
```

Para la ejecución de las pruebas necesitábamos conocer que IP se había asignado a cada una de las MV, para ello habíamos instalado la aplicación *arpscan*, que nos permitió recopilar las direcciones IP de todas las máquinas conectadas a la interfaz virbr0:

```
$ sudo arp-scan --interface=virbr0 192.168.122.0/24
Interface: virbr0, datalink type: EN10MB (Ethernet)
Starting arp-scan 1.7 with 256 hosts (http://www.nta-monitor.com/tools/arp\_scan/)
192.168.122.IP1    de:ad:be:ef:13:96    (Unknown)
192.168.122.IP2    de:ad:be:ef:41:41    (Unknown)
192.168.122.IP2    de:ad:be:ef:22:92    (Unknown)
192.168.122.IP4    de:ad:be:ef:08:60    (Unknown)
```

De esta manera, conocíamos las direcciones IP de las MV y podíamos comenzar a lanzar la primera prueba empleando el script para http, que se ha descrito anteriormente. No era necesario, en este caso, realizar ningún cambio en el espacio reservado para swap, ya que la máquina se encontraba inicialmente así preparada. Por lo tanto, a continuación iniciamos el script del potenciómetro y lanzamos las pruebas con el script para http.

```
$ sudo /tmp/powermeter/powermeter.sh start

$ sudo /home/raul/scripts/httpperfVM.sh IP1 IP2 IP3 IP4
```

Los resultados tanto de la potencia consumida como de httpperf se exponen en el apartado relativo a resultados.

- HDD + SSD swap

Volvemos en este caso a lanzar la prueba empleando las mismas MV clonadas en el apartado anterior, sin embargo, desactivamos el swap en HDD y lo activamos en el disco SSD, para ello ejecutamos:

```
$ sudo swapoff /dev/mapper/vg_prats-lv_swap
$ sudo swapon /dev/sdb2
```

Y pudimos comprobarlo ejecutando:

```
$ cat /proc/swaps
Filename                                Type           Size           Used           Priority
/dev/sdb2                               partition     59898348       0              -1
```

Con el swap ya el disco SSD procedimos a lanzar de nuevo las pruebas tras iniciar el potenciómetro como hemos hecho en el apartado anterior.

- SSD + SSD swap

Antes de comenzar con la siguiente prueba debíamos situar las MV en el disco SSD. Para ello en primer lugar destruimos las MV que habíamos creado en el HDD:

```
$ sudo /home/raul/scripts/./tfc.sh destroy /media/hdd/httpImages/ vmHttp
```

El siguiente paso fue montar el disco SSD y clonar unas nuevas MV en esta situación, y por lo tanto, editamos mínimamente la ejecución del script con el nuevo directorio de clonado:

```
$ sudo /home/raul/scripts/./tfc.sh clone /media/ssd/httpImages/ vmHttp
```

Una vez clonadas las máquinas de nuevo procedimos a iniciarlas y averiguar sus direcciones IP como hicimos en el apartado relativo a HDD.

```
$ sudo /home/raul/scripts/./tfc.sh start vmHttp
```

```
$ sudo arp-scan --interface=virbr0 192.168.122.0/24
```

Con las direcciones IP de cada MV ya podíamos iniciar el potenciómetro y ejecutar el script que hemos preparado para estas pruebas.

- Iodrive + SSD swap

El procedimiento ha sido básicamente el mismo, previa destrucción de las MV con nuestro script, clonamos las máquinas en el iodrive, y a continuación las iniciamos. Una vez que se estuvieron ejecutando las MV, procedimos a averiguar sus direcciones IP con *arp-scan*, y a ejecutar la prueba tras previamente haber iniciado el potenciómetro. En esta ocasión, y como el swap ya estaba preparado en el disco SSD, no tuvimos que hacer cambios.

b) Sysbench

- HDD + HDD swap

Antes de comenzar clonamos la MV semilla del servidor de BD en el disco HDD e iniciamos sus clones:

```
$ sudo /home/raul/scripts/./tfc.sh clone /media/hdd/sqlImages/ vmSQL
```

```
$ sudo /home/raul/scripts/./tfc.sh start vmSQL
```

Raúl López Martín

Una vez las máquinas se encontraban funcionando, al igual que en las pruebas anteriores, ejecutamos la utilidad arp-scan para conocer las direcciones IP de las MV sobre las que efectuaríamos las pruebas.

```
$ sudo arp-scan --interface=virbr0 192.168.122.0/24
```

Estos eran los hosts que pasaríamos a sysbench. Para comenzar la prueba primero preparamos la base de datos de cada MV creando las tablas sobre las que trabajaremos, para ello ejecutamos una vez por cada máquina la instrucción:

```
$ sysbench --test=oltp --db-driver=mysql --mysql-user=sqluser --mysql-  
password="secretWord" --mysql-db=sbtest --mysql-host=192.168.122.IP --mysql-  
port=3306 --mysql-table-engine=innodb prepare
```

Puesto que evaluaríamos todas las máquinas conjuntamente empleamos la opción *mysql-host*, que nos permite pasar a sysbench una lista separada por comas de todas las MV, de esta manera las conexiones se distribuyen entre todas siguiendo una lógica *round-robin*. Este test y sus parámetros se seleccionaron, a través de pruebas de ensayo y error, para tener una duración aproximada de unos 4 minutos y generar un uso intensivo de la BD. Previamente a la ejecución de la prueba iniciamos el potenciómetro para medir los consumos producidos por la actividad:

```
$ sysbench --max-requests=1000 --test=oltp --oltp-index-updates=25 --oltp-distinct-  
ranges=25 --oltp-sum-ranges=25 --db-driver=mysql --mysql-user=sqluser --mysql-  
password="secretWord" --mysql-db=sbtest --mysql-  
host=192.168.122.IP1,192.168.122.IP2,192.168.122.IP3,192.168.122.IP4 --mysql-  
port=3306 --mysql-table-engine=innodb --num-threads=400 run
```

Para finalizar, eliminamos las tablas que empleamos para realizar las pruebas y de nuevo ejecutamos una vez por cada MV:

```
$ sysbench --test=oltp --db-driver=mysql --mysql-user=sqluser --mysql-  
password="secretWord" --mysql-db=sbtest --mysql-host=192.168.122.IP --mysql-  
port=3306 cleanup  
sysbench 0.4.12: multi-threaded system evaluation benchmark  
Dropping table 'sbtest'..  
Done.
```

- HDD + SSD swap

Desactivamos swap en HDD y lo activamos en el SSD como hicimos con las pruebas del servidor web, omitimos el comando por tratarse del mismo en este caso.

Una vez configurado el swap los comandos a ejecutar son exactamente los mismos que hemos realizado en la prueba anterior: preparar la base de datos, iniciar potenciómetro, ejecutar sysbench run, limpiar la base de datos y parar el potenciómetro.

- SSD + SSD swap

Repetimos la secuencia una vez mas, primero de todo eliminamos las máquinas del disco magnético y las clonamos en el SSD. Para ello, de nuevo empleamos nuestro script tal como hicimos con httpperf.

```
$ sudo /home/raul/scripts/.tfc.sh destroy /media/hdd/sqlImages/ vmSQL
$ sudo /home/raul/scripts/.tfc.sh clone /media/ssd/sqlImages/ vmSQL
$ sudo /home/raul/scripts/.tfc.sh start vmSQL
```

Una vez hecho esto las pruebas a ejecutar fueron las mismas y los comandos por lo tanto réplica de los ya ejecutados. Los resultados obtenidos pueden consultarse mas adelante.

- Iodrive + SSD swap

Mismos pasos que las veces anteriores pero clonando sobre la unidad iodrive.

c) b_eff_io

Para la ejecución de las pruebas con b_eff_io el procedimiento difiere bastante del empleado con el resto de herramientas de *benchmarking*. Para empezar, debido a la necesidad de emplear direcciones IP estáticas no clonamos, como hemos hecho en los otros casos, sobre los diferentes medios *hardware*.

Lo que hicimos fue ejecutar las pruebas sobre cada uno de los discos, y posteriormente, editar los archivos xml que definen las MV para especificar una nueva ubicación para sus imágenes. Empleando el editor nano para editar los xml mencionados ejecutamos:

```
$ nano /etc/libvirt/qemu/MV.xml
```

Una vez hecho esto ya procedimos a mover las imágenes al nuevo directorio que indicamos en el xml, así por ejemplo, para pasar del disco HDD al SSD:

```
$ sudo mv /media/hdd/MPIImages/MV.img /media/ssd/MPIImages/MV.img
```

Una vez movidas las imágenes procedimos a reiniciar los servicios de libvirt para que los cambios se hicieran efectivos con total seguridad:

```
$ sudo /etc/init.d/libvirtd restart  
$ sudo /etc/init.d/libvirt-guest restart
```

Por cada situación a comprobar arrancamos las MV empleando el script e iniciamos con el cada uno de los nodos del cluster que hemos preparado. Previamente habíamos iniciado la MV que funcionaba como servidor de nfs para que todas las máquinas nodo montarán la partición nfs al iniciarse.

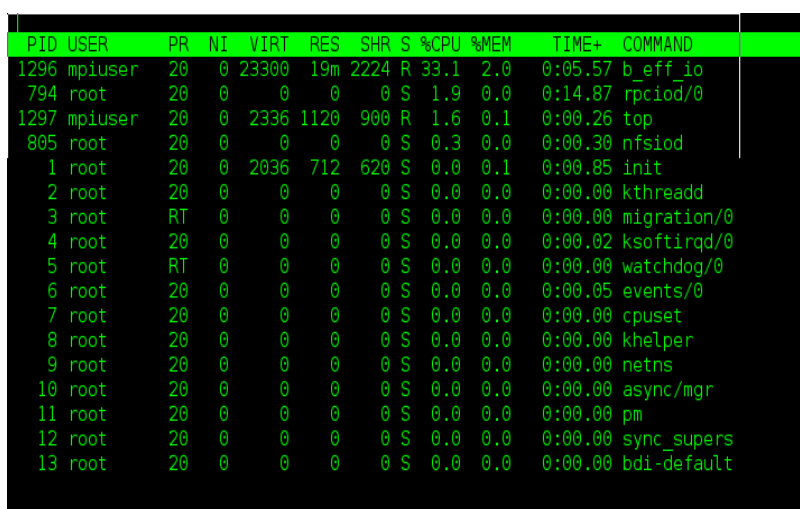
Como parámetros para la ejecución seleccionamos: 1024 MB de memoria por nodo y 4096 MB de memoria total para 4 procesos. La elección del número de procesos vino motivada por estar trabajando en una máquina de 4 núcleos y con 4 GB de memoria RAM, esperábamos que con estos parámetros se llegase a llenar la memoria y fuera necesario emplear swap durante la ejecución de las pruebas.

Pese a la cautela ninguna ejecución de b_eff_io con estas características termino en un tiempo razonable, tanto empleando mpich2 como con una segunda configuración del cluster que realizamos empleando openmpi. En un principio supusimos que el motivo podría ser un aparente bug en la implementación de nfs para Linux²⁵, que provoca que nfs se bloquee por procesos rcpiod que ocupan la CPU, provocando un bucle sin realizar en realidad ningún tipo de tarea.

Optamos por lo tanto, por monitorizar la herramienta vigilando la aparición de este tipo de procesos y eliminándolos para evitar los supuestos bucles.

Para ello confeccionamos un pequeño script que buscaba estos procesos rcpiod y los eliminaba.

Sin embargo, el comportamiento de las pruebas continuo en los mismos términos. Si



PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
1296	mpiuser	20	0	23300	19m	2224	R	33.1	2.0	0:05.57	b_eff_io
794	root	20	0	0	0	0	S	1.9	0.0	0:14.87	rcpiod/0
1297	mpiuser	20	0	2336	1120	900	R	1.6	0.1	0:00.26	top
805	root	20	0	0	0	0	S	0.3	0.0	0:00.30	nfsiod
1	root	20	0	2036	712	620	S	0.0	0.1	0:00.85	init
2	root	20	0	0	0	0	S	0.0	0.0	0:00.00	kthread
3	root	RT	0	0	0	0	S	0.0	0.0	0:00.00	migration/0
4	root	20	0	0	0	0	S	0.0	0.0	0:00.02	ksoftirqd/0
5	root	RT	0	0	0	0	S	0.0	0.0	0:00.00	watchdog/0
6	root	20	0	0	0	0	S	0.0	0.0	0:00.05	events/0
7	root	20	0	0	0	0	S	0.0	0.0	0:00.00	cpuset
8	root	20	0	0	0	0	S	0.0	0.0	0:00.00	khelper
9	root	20	0	0	0	0	S	0.0	0.0	0:00.00	netns
10	root	20	0	0	0	0	S	0.0	0.0	0:00.00	async/mgr
11	root	20	0	0	0	0	S	0.0	0.0	0:00.00	pm
12	root	20	0	0	0	0	S	0.0	0.0	0:00.00	sync_supers
13	root	20	0	0	0	0	S	0.0	0.0	0:00.00	bdi-default

²⁵ Según referencia encontrada ver [33]

detectamos que el problema era mas grave en unos discos que en otros, concretamente sobre el disco magnético la ejecución solo finalizaba con un único proceso.

Como comentábamos en la explicación sobre `b_eff_io` esta herramienta escribe sobre un directorio una serie de datos y sobre estos realiza una serie de operaciones a partir de las cuales genera su resultado. Decidimos entonces vigilar detenidamente la ejecución de la herramienta sobre el directorio que comprueba.

Al hacer este seguimiento detectamos que el test paraba habitualmente en algunos de los archivos generados y que al hacerlo el proceso `rcpiod` desaparece. Si cuando esto sucede procedemos a borrar el archivo en cuestión el proceso `rcpiod` recupera la CPU y la ejecución continua, por lo que el comportamiento observado por nosotros, al menos aparentemente, es contrario a la información encontrada^[33].

Debe tenerse en cuenta que la ejecución de las pruebas se ha realizado en todos los casos con al menos un borrado forzado durante su ejecución. Según hemos podido detectar ha causado que el número de errores reportado por `b_eff_io` sea tal vez mayor, y que en algunas ocasiones picos anómalos en las velocidades de lectura/escritura. En estos casos se ha repetido nuevamente el experimento.

El comando exacto que hemos empleado para lanzar las pruebas se muestra a continuación. La última parte del comando se ha utilizado para permitir lanzar la ejecución de `b_eff_io` en fondo si error. Los posibles mensajes de error se recogían en `results.out` y eran comprobados al realizar la ejecución:

```
$ mpirun -np 4 --hostfile /mirror/.mpi_hostfile ./b_eff_io -MB 1024 -MT 4096 -T 600 -p /mirror/scratch -f /mirror/b_eff_io/b_eff_io_iodrive_N4 </dev/null &> results.out &
```

La ejecución del comando la hemos realizado sobre el home de `mpiuser`, en el cual teníamos exactamente en la misma ubicación el ejecutable de `b_eff_io`, tanto los resultados como los errores se guardaron en la directorio `nfs`.

Dado que la ejecución de las pruebas requería de una observación continuada, procedimos a evaluar únicamente el consumo en cada entorno de hardware durante un intervalo de 600 segundos. Durante este intervalos se realizaron operaciones intensivas de lectura y escritura por todos los nodos con `b_eff_io`. Los resultados obtenidos los exponemos en el apartado de resultados.

4.7. Otras consideraciones

Durante la preparación del entorno para este proyecto han sido varios los obstáculos que se han debido solventar. Estos obstáculos pueden clasificarse principalmente en dos: primero problemas de permisos y segundo problemas de compatibilidad entre los diversos elementos que se han tratado de combinar.

El primer obstáculo surgió tras la instalación de KVM, que en un primer momento se realizó sobre una distribución Scientific Linux Carbon. La instalación de KVM no supuso ningún problema, si bien la ejecución de virt-install a través de ssh resultó una pesadilla en cuanto a permisos de ejecución. A continuación aportamos uno de tantos errores obtenidos en esta fase de la preparación del entorno como muestra.

```
> [Errno 13] Permiso denegado: '/directory/subdirectiry/rlopez/.virtinst 'Traceback (most recent call last):
> File "/usr/bin/virt-install", line 1004, in sys.exit(main())
> File "/usr/bin/virt-install", line 973, in main cli.setupLogging("virt-install", options.debug, options.quiet) File "/usr/lib/python2.6/site-packages/virtinst/cli.py", line 165, in setupLogging os.mkdir(vi_dir, 0751)
> OS Error: [Errno 13] Permiso denegado: '/directory/subdirectiry/rlopez/.virtinst
```

Para tratar de solventar este inconveniente fueron varias las soluciones que se manejaron: problemas de permisos en la ubicación de las imágenes de las máquinas virtuales, problemas con el driver de conexión (KVM, qemu), soporte de virtualización desactivado .. etc. Finalmente no fue posible localizar el problema y se optó por reinstalar completamente el SO lo que supuso un primer retraso inicial.

Aprovechando que el sistema se instalaba nuevamente optamos por emplear una distribución Debian Squeeze, que había funcionado en una máquina de menores características. Lamentablemente, una vez instalado el sistema cuando solo quedaba pendiente la instalación de los drivers del potenciómetro y el iodrivo, vimos que no existía un driver disponible para Debian Squeeze y la primera versión de iodrivo.

Se optó finalmente por emplear una distribución Fedora 17 para que la que comprobamos que si existía driver. Por desgracia una vez más se chocó contra un nuevo escollo, en este caso se trataba de un parche^[26] de nueva aplicación en el kernel que modificaba la estructura de struct pci_bus en pci.h. Pasando de la versión:

```
unsigned char secondary; /* number of secondary bridge */
unsigned char subordinate; /* max number of subordinate buses */
```

A una nueva versión en la que las estructuras anteriores desaparecen:

```
struct resource busn_res; /* bus numbers routed to this bus */
```

De esta manera, cuando el driver trataba de inicializar *subordinate* y *secondary* fallaba y no finalizaba el empaquetado. De acuerdo a la información existente en el parche tratamos de modificar y empaquetar el driver de iodrive empleando *bus_res* para sustituir a *subordinate* y *secondary*, consiguiendo de esta manera compilar e instalar el driver. Sin embargo, no fue posible posteriormente ejecutar el comando que permitía emplear iodrive como una unidad de disco (*fiio-attach*), así que finalmente se optó por realizar una consulta al soporte de Fusion-io.

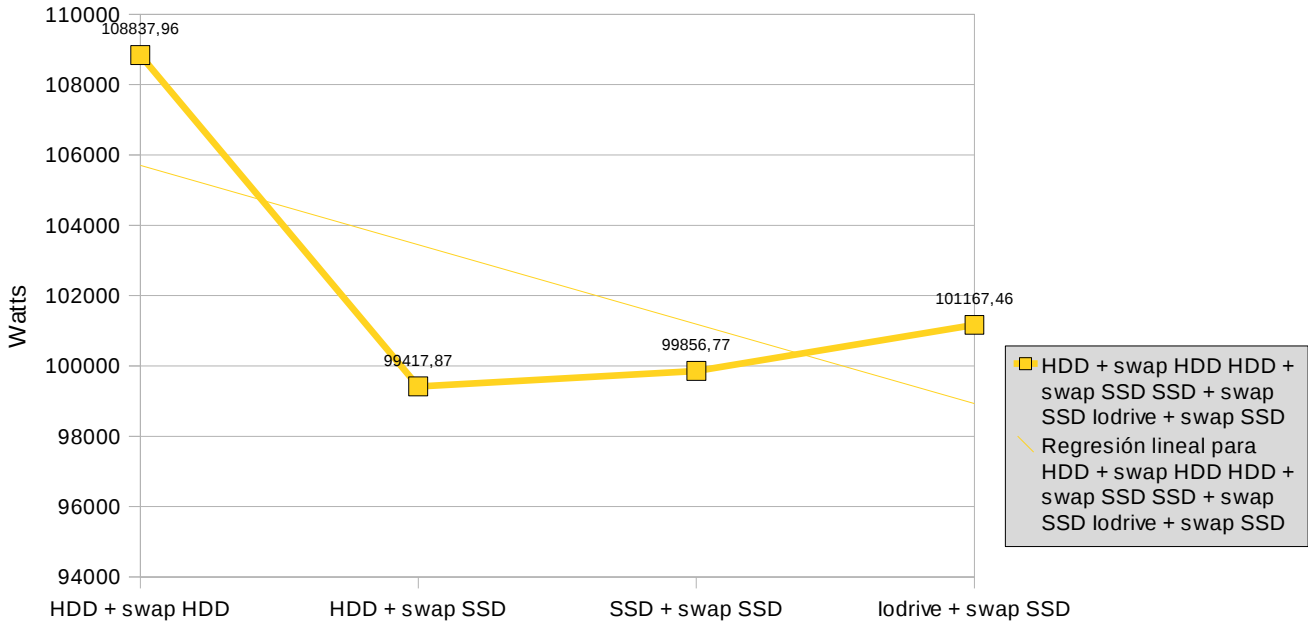
Cuando obtuvimos respuesta del soporte de Fusion-io nos indicaron que procediéramos a actualizar el firmware del producto. Esto se realizó con el siguiente comando de iodrive:

```
$ fiio-update-iodrive
```

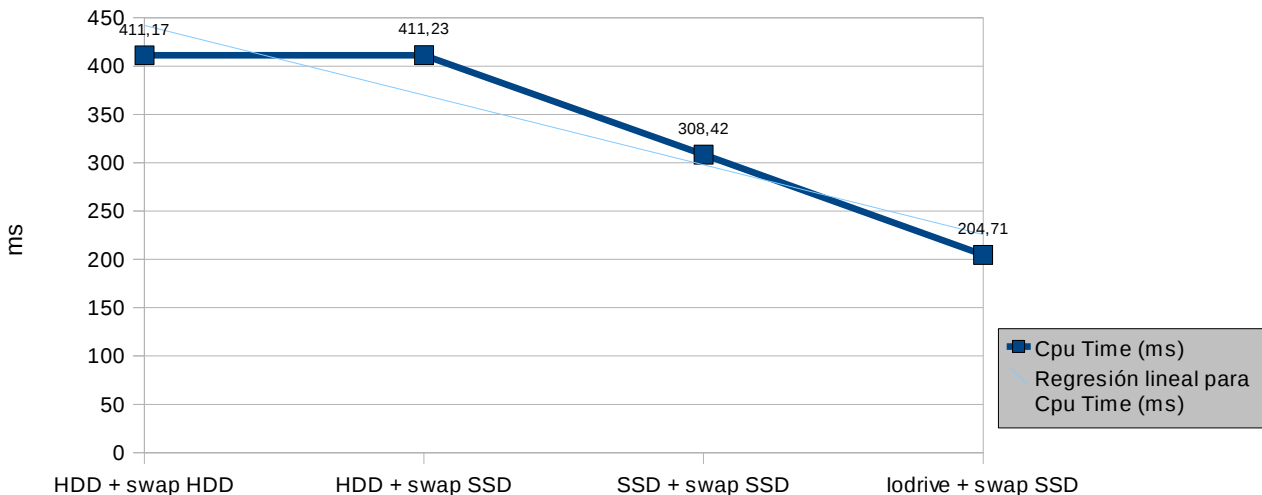
Una vez actualizado el firmware y reiniciado el equipo se ejecutó *fiio-attach* automáticamente, solo nos restó montar un sistema de ficheros sobre una de ellas y comenzar las pruebas que ya habíamos realizado sobre los otros medios.

5. Resultados y conclusiones

5.1. httperf



Como puede verse en los resultados aportados en el anexo 1 de este documento, el funcionamiento de las MV para servicios web en función de la tecnología de almacenamiento empleada ha sido bastante semejante. Si podemos destacar algún aspecto sería sin duda la caída en el consumo una vez hemos empleado el disco SSD como swap y un pico en el consumo al emplear el lodrive para servicio web. En cuanto al rendimiento, la mejora es mas clara.



Podemos ver en el gráfico que al abandonar los medios con partes móviles mecánicas, el uso de CPU por operación ha disminuido notablemente. Un menor uso de este recurso, proporciona mayor velocidad en la ejecución y permite aprovechar este tiempo para

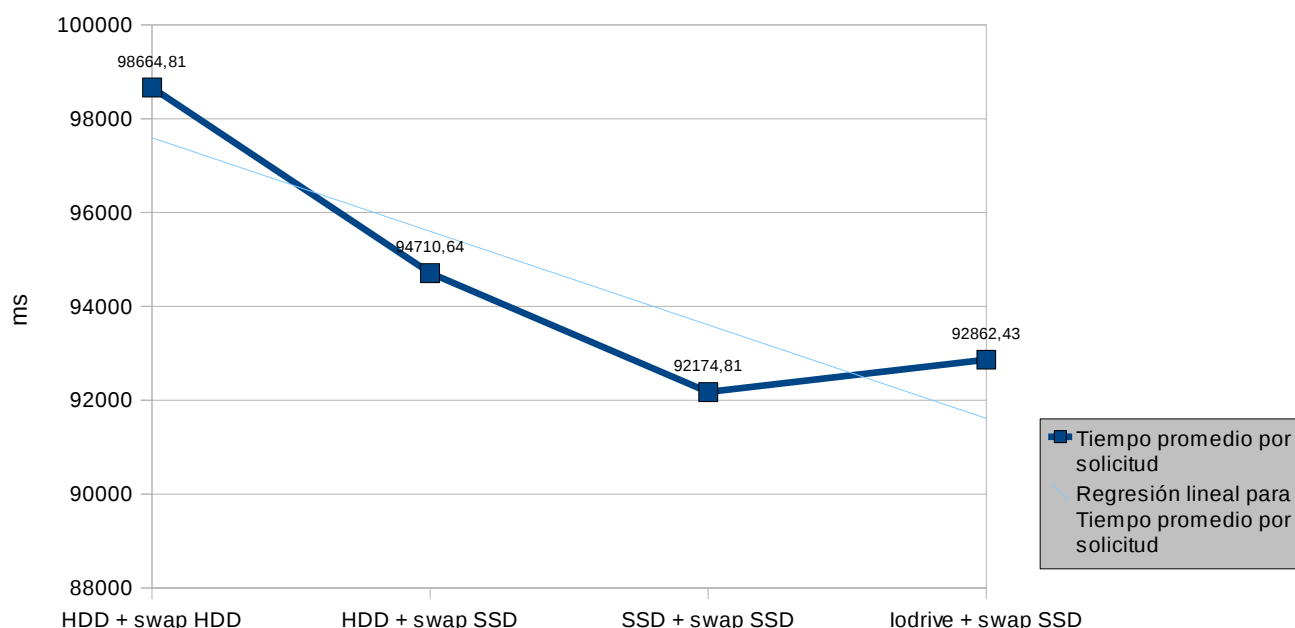
atender otras solicitudes al servidor.

En definitiva vemos, basándonos en este parámetro, que el rendimiento mejora. Esta mejora, discos magnéticos frente a nuevas tecnologías, es muy alta y mas aún si lo comparamos con el iodrivo. Además, el rendimiento del disco SSD frente al iodrivo es de aproximadamente 104 ms, de lo que podemos intuir que el uso de tecnologías específicas para NAND supone también una mejoría.

5.2. sysbench

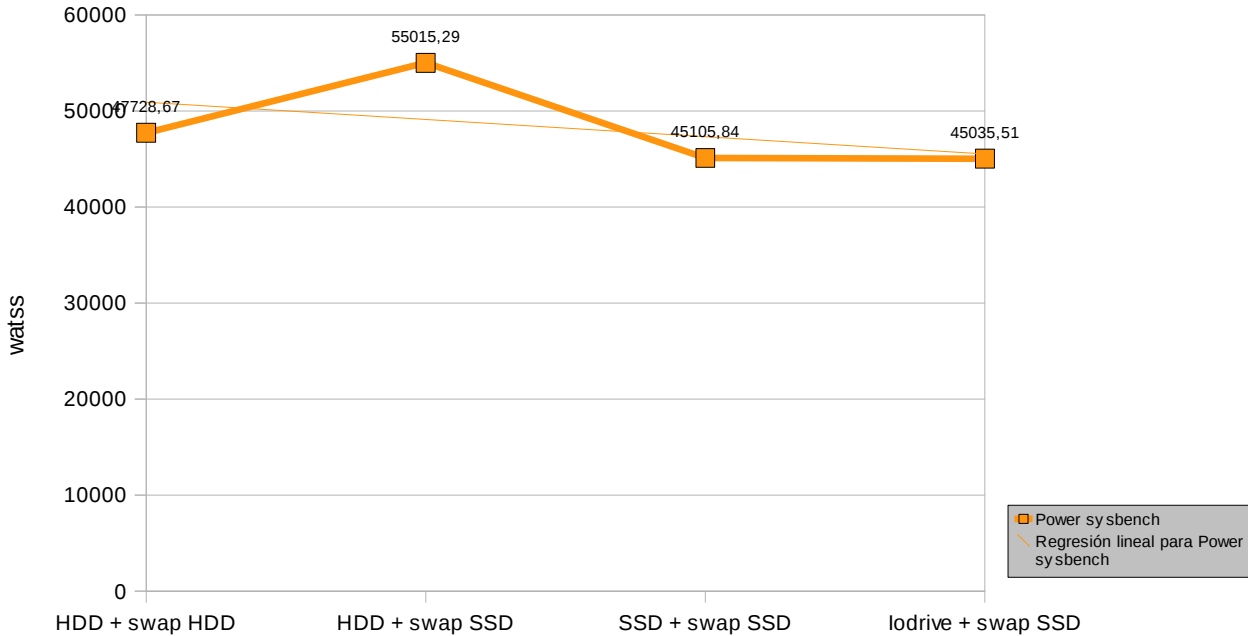
En los resultados obtenidos con sysbench, si evaluamos por ejemplo el tiempo medio utilizado por cada solicitud a la base de datos, vemos que la mejora es clara con unas tecnologías frente a otras y con cada prueba el tiempo se ha reducido.

La tendencia, si observamos la gráfica a continuación, es a la baja con cada subsiguiente prueba y mayor conforme abandonamos HDD. La diferencia, sin embargo, entre SSD y el iodrivo es de 700 ms a favor del primero. Si comparamos estos datos con los obtenidos para servidor web, donde el rendimiento de iodrivo fue mejor, suponemos que se trate de una ejecución anecdótica y sería conveniente realizar nuevas pruebas, que lamentablemente, no podemos hacer por falta de tiempo.

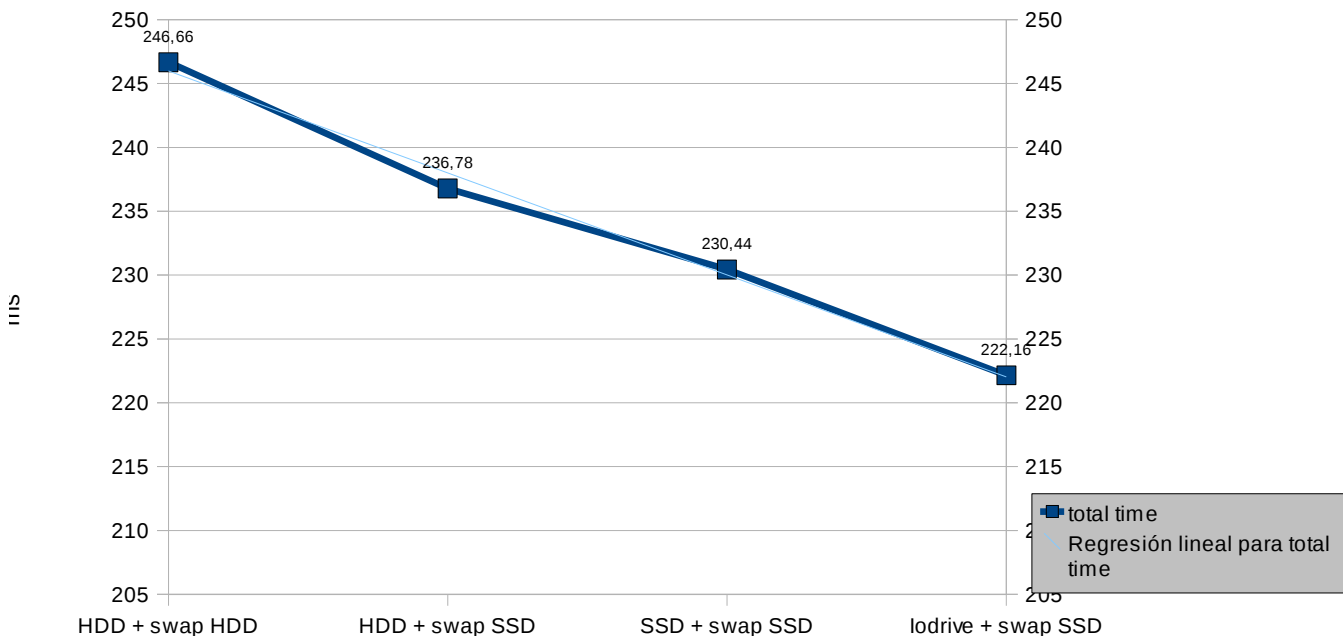


En cuanto al consumo, en el caso de esta prueba, las diferentes medidas han diferido mínimamente. Tal vez la BD si no es muy extensa, y en este caso no lo es, se encuentre cargada en memoria produciendo pocas lecturas a disco y por lo tanto el consumo no haya dependido del disco utilizado. Si nos fijamos en el gráfico de la página siguiente, la mayor diferencia de consumo se produce con el cambio de la memoria swap al disco SSD, y se

mantiene muy cercana en valores cuando comparamos el disco SSD con iodrive. El uso, por otro lado, del disco HDD con swap alojado en SSD ha supuesto el mayor consumo de todos los medidos. Seria conveniente repetir esta prueba para determinar si se trata de un caso puntual, o bien, este comportamiento se reproduce todas las veces.



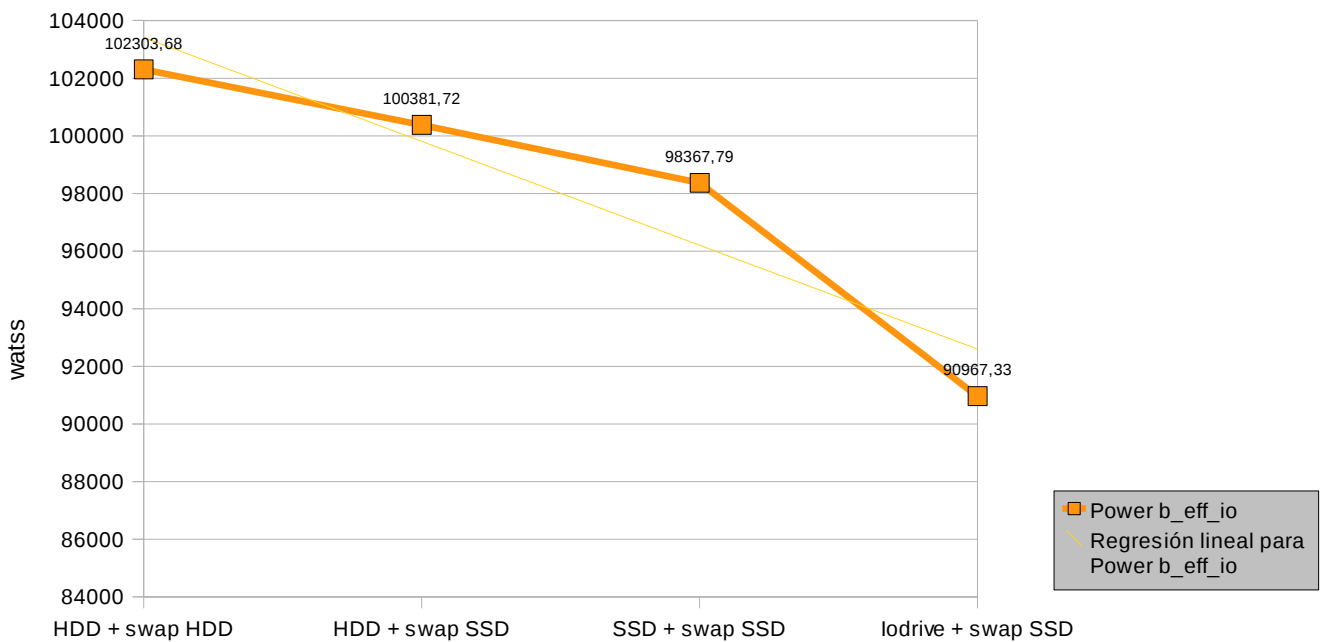
Lo que si es claro es que si comparamos los tiempos, que en este caso los proporciona la propia herramienta, para una misma batería de operaciones estos se han reducido notablemente en cada prueba. Por lo que pese a un mismo consumo, o al menos muy aproximado, si hemos ganado en velocidad de ejecución. Viendo que el tiempo medio por solicitud de iodrive ha sido mas alto, sería interesante evaluar la causa de que finalmente la ejecución sobre el disco SSD haya sido mas lenta.



5.3. b_eff_io

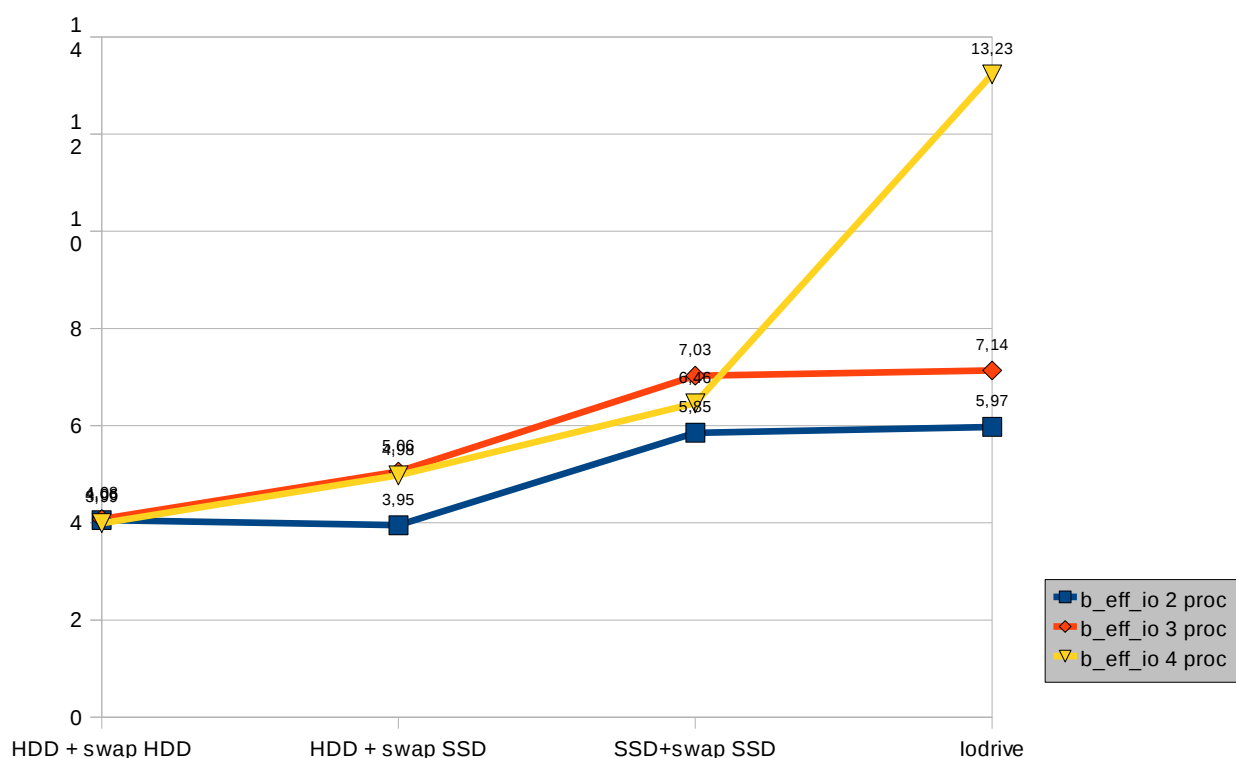
De todas las medidas obtenidas en las pruebas, los resultados de esta son los mas claros en cuanto a consumo de energía y rendimiento. Si valoramos la tendencia de las gráficas vemos claramente que conforme abandonamos los discos magnéticos frente a tecnologías NAND la mejora es cada vez mayor.

Pero además si evaluamos la diferencia en el consumo que supone el empleo del disco SSD, empleando tecnologías de conexión pensadas para discos magnéticos, frente a la unidad Iodrive podemos ver como supone casi 7500 watios de diferencia, la mayor pendiente entre dos valores de todas la pruebas realizadas, y realizando un calculo aproximado 45 kw/h de diferencia en el consumo.



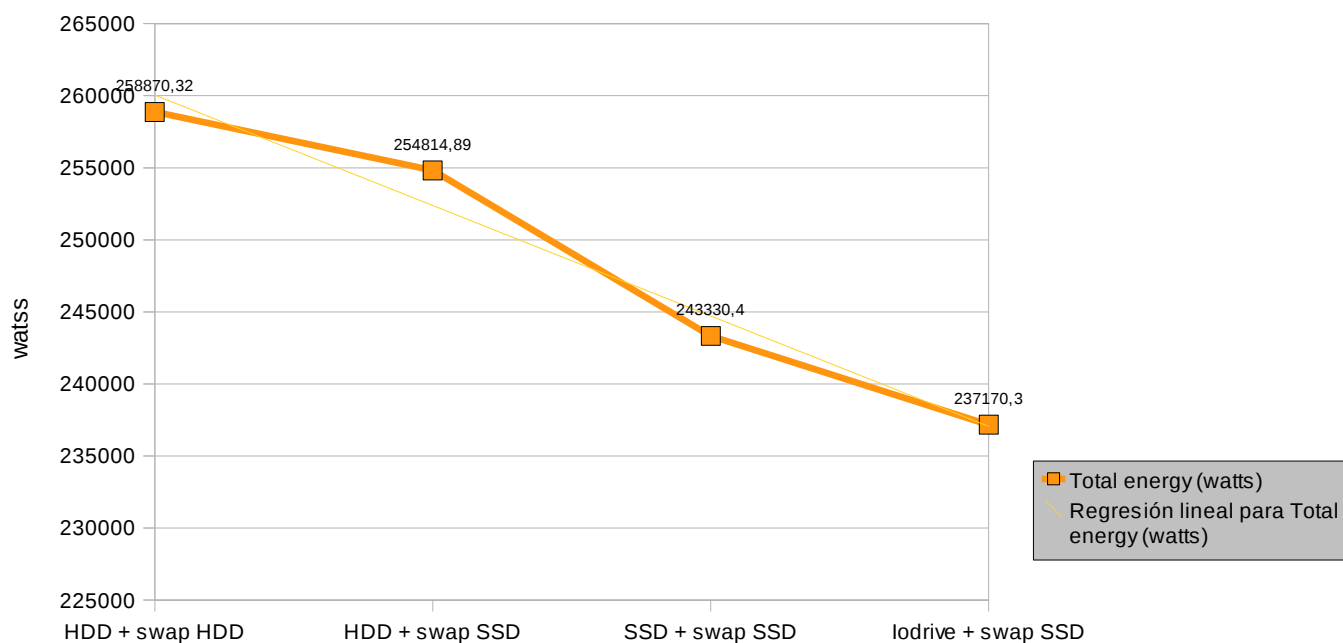
En cuanto a los valores `b_eff_io`, en la gráfica de la página siguiente presentamos los resultados obtenidos para desde dos hasta cuatro procesos. Estos valores los hemos obtenido con la prueba descrita anteriormente. Con esta prueba, en la que las operaciones de lectura/escritura son mas intensivas, la ventaja de iodrive es muy clara frente a las otras tecnologías y supone un incremento en la velocidad realmente importante.

Es, en consecuencia, cuando se emplean los discos para funciones que requieren una gran cantidad de acceso a disco, donde tecnologías de este tipo pueden resultar amortizadas a mas corto plazo que frente a otras. Podemos valorar además, remitiéndonos a los valores obtenidos en nuestras pruebas, que esta mejora se ha mantenido en todos los casos independientemente del número de procesos.



5.4. Discusión

Si observamos en conjunto el total de los consumos obtenidos de todas las pruebas (ver gráfico) en cada una de ellas el consumo ha sido menor y en ocasiones incluso el rendimiento ha salido beneficiado, con un menor consumo.



Resulta pues claro que pese a que aun a día de hoy las nuevas tecnologías de almacenamiento suponen un sobre coste frente a los a por ejemplo los discos magnéticos, el uso de estas nuevas tecnologías puede quedar amortizado si consideramos los ahorros energéticos y de rendimiento que supone su uso.

Frente a futuras nuevas pruebas sería interesante: realizar mediciones empleando un sistema distinto a nfs (ocfs, lutre, cfs ...) y ver cual es el comportamiento de cada uno de ellos, inclusive tratar de emplear otra distribución Linux y comprobar si versiones diferentes de nfs sufren el mismo tipo de problemas que la empleada. También nos ha quedado pendiente, por falta de tiempo, valorar el rendimiento de iodrive cuando es usado como disco y como swap, ya que debido a la dificultad para llegar a tenerlo en funcionamiento no nos ha sido posible emplear toda su capacidad, y únicamente hemos utilizado una mínima parte.

5.5. Trabajo futuro

Tras el presente proyecto son muchas las opciones que se han quedado sin abordar, y es que el número de combinaciones posibles con los medios disponibles era alto. Nos ha quedado pendiente, por ejemplo, comprobar la eficacia de iodrive como swap, lo que habría sido interesante sobre todo en los casos en los que la tecnología de almacenamiento utilizada para swap ha sido aparentemente determinante.

Los problemas surgidos con la utilización de nfs nos limitaron a la hora de emplear un número mayor de MV, por lo que otra opción interesante sería repetir el proyecto en un entorno que nos permitiese usar un mayor número de MV para las pruebas. Otra opción de interés habría sido situar las MV en la memoria swap para evaluar como varia su rendimiento en este caso.

En cuanto a las aplicaciones nos quedó pendiente evaluar el rendimiento con el funcionamiento de aplicaciones diferentes de las de benchmarking, como por ejemplo aplicaciones matemáticas de factorización de números primos, que nos acercarán aun mas al funcionamiento de sistemas productivos reales.

También nos hubiera gustado poder realizar varias baterías de pruebas con cada uno de los entornos planteados, realizando un número alto de ejecuciones y obteniendo una media de sus resultados, pues nos habría permitido eliminar resultados espurios con total certeza. Otra opción habría sido emplear diferentes herramientas de benchmarking, que nos ofrecieran otra aproximación al rendimiento, facilitándonos una comparación desde las diferentes aproximaciones utilizadas por sus desarrolladores.

6. Anexos

6.1. Anexo 1

El presente anexo recoge los resultados obtenidos de cada una de las pruebas. Estos resultados, a excepción de `b_eff_io`, se presentan en su totalidad tal y como fueron obtenidos de las diferentes herramientas empleadas para el benchmarking.

a) httperf

- HDD + swap HDD

- Datos potenciómetro:

```
$ sudo /tmp/powermeter/powermeter.sh totalenergy  
108837.962000
```

```
$ sudo /tmp/powermeter/powermeter.sh totaltime  
656.8430
```

- *MV 0:*

```
httperf --hog --client=0/1 --server=192.168.122.148 --port=80 --uri=/ --send-  
buffer=4096 --recv-buffer=16384 --wsess=$  
Maximum connect burst length: 1  
Total: connections 4 requests 60 replies 60 test-duration 560.076 s  
Connection rate: 0.0 conn/s (140018.9 ms/conn, <=2 concurrent connections)  
Connection time [ms]: min 140015.5 avg 140018.9 max 140023.6 median 0.0 stddev 4.0  
Connection time [ms]: connect 1.8  
Connection length [replies/conn]: 15.000  
Request rate: 0.1 req/s (9334.6 ms/req)  
Request size [B]: 68.0  
Reply rate [replies/s]: min 0.0 avg 0.1 max 0.6 stddev 0.1 (112 samples)  
Reply time [ms]: response 1.3 transfer 0.0  
Reply size [B]: header 283.0 content 177.0 footer 0.0 (total 460.0)  
Reply status: 1xx=0 2xx=60 3xx=0 4xx=0 5xx=0  
CPU time [s]: user 142.92 system 411.03 (user 25.5% system 73.4% total 98.9%)  
Net I/O: 0.1 KB/s (0.0*10^6 bps)  
Errors: total 0 client-timo 0 socket-timo 0 connrefused 0 connreset 0  
Errors: fd-unavail 0 addrunavail 0 ftab-full 0 other 0  
Session rate [sess/s]: min 0.00 avg 0.01 max 0.20 stddev 0.03 (4/4)  
Session: avg 1.00 connections/session  
Session lifetime [s]: 140.0  
Session failtime [s]: 0.0  
Session length histogram: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 4
```

- **MV 1:**

```
httperf --hog --client=0/1 --server=192.168.122.224 --port=80 --uri=/ --send-  
buffer=4096 --recv-buffer=16384 --wsess=$  
Maximum connect burst length: 1  
Total: connections 4 requests 60 replies 60 test-duration 560.053 s  
Connection rate: 0.0 conn/s (140013.2 ms/conn, <=2 concurrent connections)  
Connection time [ms]: min 140007.1 avg 140013.2 max 140023.2 median 0.0 stddev 7.0  
Connection time [ms]: connect 2.0  
Connection length [replies/conn]: 15.000  
Request rate: 0.1 req/s (9334.2 ms/req)  
Request size [B]: 68.0  
Reply rate [replies/s]: min 0.0 avg 0.1 max 0.6 stddev 0.1 (112 samples)  
Reply time [ms]: response 0.9 transfer 0.0  
Reply size [B]: header 283.0 content 177.0 footer 0.0 (total 460.0)  
Reply status: 1xx=0 2xx=60 3xx=0 4xx=0 5xx=0  
CPU time [s]: user 146.19 system 411.11 (user 26.1% system 73.4% total 99.5%)  
Net I/O: 0.1 KB/s (0.0*10^6 bps)  
Errors: total 0 client-timo 0 socket-timo 0 connrefused 0 connreset 0  
Errors: fd-unavail 0 addrunavail 0 ftab-full 0 other 0  
Session rate [sess/s]: min 0.00 avg 0.01 max 0.20 stddev 0.03 (4/4)  
Session: avg 1.00 connections/session  
Session lifetime [s]: 140.0  
Session failtime [s]: 0.0  
Session length histogram: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 4
```

- **MV 2:**

```
httperf --hog --client=0/1 --server=192.168.122.39 --port=80 --uri=/ --send-buffer=4096  
--recv-buffer=16384 --wsess=4$  
Maximum connect burst length: 1  
Total: connections 4 requests 60 replies 60 test-duration 560.062 s  
Connection rate: 0.0 conn/s (140015.6 ms/conn, <=2 concurrent connections)  
Connection time [ms]: min 140010.8 avg 140015.6 max 140021.3 median 0.0 stddev 4.5  
Connection time [ms]: connect 1.7  
Connection length [replies/conn]: 15.000  
Request rate: 0.1 req/s (9334.4 ms/req)  
Request size [B]: 67.0  
Reply rate [replies/s]: min 0.0 avg 0.1 max 0.6 stddev 0.1 (112 samples)  
Reply time [ms]: response 1.1 transfer 0.0  
Reply size [B]: header 283.0 content 177.0 footer 0.0 (total 460.0)  
Reply status: 1xx=0 2xx=60 3xx=0 4xx=0 5xx=0  
CPU time [s]: user 145.37 system 412.13 (user 26.0% system 73.6% total 99.5%)  
Net I/O: 0.1 KB/s (0.0*10^6 bps)  
Errors: total 0 client-timo 0 socket-timo 0 connrefused 0 connreset 0  
Errors: fd-unavail 0 addrunavail 0 ftab-full 0 other 0
```

Raúl López Martín

Session rate [sess/s]: min 0.00 avg 0.01 max 0.20 stddev 0.03 (4/4)
Session: avg 1.00 connections/session
Session lifetime [s]: 140.0
Session failtime [s]: 0.0
Session length histogram: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 4

- **MV 3:**

```
htperf --hog --client=0/1 --server=192.168.122.53 --port=80 --uri=/ --send-buffer=4096
--recv-buffer=16384 --wsess=4$
Maximum connect burst length: 1
Total: connections 4 requests 60 replies 60 test-duration 560.024 s
Connection rate: 0.0 conn/s (140006.0 ms/conn, <=2 concurrent connections)
Connection time [ms]: min 140005.8 avg 140006.0 max 140006.4 median 0.0 stddev 0.3
Connection time [ms]: connect 0.8
Connection length [replies/conn]: 15.000
Request rate: 0.1 req/s (9333.7 ms/req)
Request size [B]: 67.0
Reply rate [replies/s]: min 0.0 avg 0.1 max 0.6 stddev 0.1 (112 samples)
Reply time [ms]: response 0.7 transfer 0.0
Reply size [B]: header 283.0 content 177.0 footer 0.0 (total 460.0)
Reply status: 1xx=0 2xx=60 3xx=0 4xx=0 5xx=0
CPU time [s]: user 143.80 system 410.41 (user 25.7% system 73.3% total 99.0%)
Net I/O: 0.1 KB/s (0.0*10^6 bps)
Errors: total 0 client-timo 0 socket-timo 0 connrefused 0 connreset 0
Errors: fd-unavail 0 addrunavail 0 ftab-full 0 other 0
Session rate [sess/s]: min 0.00 avg 0.01 max 0.20 stddev 0.03 (4/4)
Session: avg 1.00 connections/session
Session lifetime [s]: 140.0
Session failtime [s]: 0.0
Session length histogram: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 4
```

- **HDD + SSD swap**

- **Datos potenciómetro:**

```
$ sudo /tmp/powermeter/powermeter.sh totalenergy
99417.874300
```

```
$ sudo /tmp/powermeter/powermeter.sh totaltime
570.2820
```

- **MV 0:**

```
htperf --hog --client=0/1 --server=192.168.122.148 --port=80 --uri=/ --send-
buffer=4096 --recv-buffer=16384 --wsess=$
Maximum connect burst length: 1
Total: connections 4 requests 60 replies 60 test-duration 560.038 s
Connection rate: 0.0 conn/s (140009.6 ms/conn, <=2 concurrent connections)
```

Raúl López Martín

Connection time [ms]: min 140005.9 avg 140009.6 max 140014.7 median 0.0 stddev 4.1
Connection time [ms]: connect 0.5
Connection length [replies/conn]: 15.000
Request rate: 0.1 req/s (9334.0 ms/req)
Request size [B]: 68.0
Reply rate [replies/s]: min 0.0 avg 0.1 max 0.6 stddev 0.1 (112 samples)
Reply time [ms]: response 0.9 transfer 0.0
Reply size [B]: header 283.0 content 177.0 footer 0.0 (total 460.0)
Reply status: 1xx=0 2xx=60 3xx=0 4xx=0 5xx=0
CPU time [s]: user 145.37 system 412.04 (user 26.0% system 73.6% total 99.5%)
Net I/O: 0.1 KB/s (0.0*10⁶ bps)
Errors: total 0 client-timo 0 socket-timo 0 connrefused 0 connreset 0
Errors: fd-unavail 0 addrunavail 0 ftab-full 0 other 0
Session rate [sess/s]: min 0.00 avg 0.01 max 0.20 stddev 0.03 (4/4)
Session: avg 1.00 connections/session
Session lifetime [s]: 140.0
Session failtime [s]: 0.0
Session length histogram: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 4

- **MV 1:**

httpperf --hog --client=0/1 --server=192.168.122.224 --port=80 --uri=/ --send-buffer=4096 --recv-buffer=16384 --wsess=\$
Maximum connect burst length: 1
Total: connections 4 requests 60 replies 60 test-duration 560.071 s
Connection rate: 0.0 conn/s (140017.7 ms/conn, <=2 concurrent connections)
Connection time [ms]: min 140013.8 avg 140017.7 max 140022.2 median 0.0 stddev 4.4
Connection time [ms]: connect 0.8
Connection length [replies/conn]: 15.000
Request rate: 0.1 req/s (9334.5 ms/req)
Request size [B]: 68.0
Reply rate [replies/s]: min 0.0 avg 0.1 max 0.6 stddev 0.1 (112 samples)
Reply time [ms]: response 1.3 transfer 0.0
Reply size [B]: header 283.0 content 177.0 footer 0.0 (total 460.0)
Reply status: 1xx=0 2xx=60 3xx=0 4xx=0 5xx=0
CPU time [s]: user 144.26 system 413.15 (user 25.8% system 73.8% total 99.5%)
Net I/O: 0.1 KB/s (0.0*10⁶ bps)
Errors: total 0 client-timo 0 socket-timo 0 connrefused 0 connreset 0
Errors: fd-unavail 0 addrunavail 0 ftab-full 0 other 0
Session rate [sess/s]: min 0.00 avg 0.01 max 0.20 stddev 0.03 (4/4)
Session: avg 1.00 connections/session
Session lifetime [s]: 140.0
Session failtime [s]: 0.0
Session length histogram: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 4

- **MV 2:**

httpperf --hog --client=0/1 --server=192.168.122.39 --port=80 --uri=/ --send-

Raúl López Martín

```
buffer=4096 --recv-buffer=16384 --wsess=4$
Maximum connect burst length: 1
Total: connections 4 requests 60 replies 60 test-duration 560.023 s
Connection rate: 0.0 conn/s (140005.7 ms/conn, <=2 concurrent connections)
Connection time [ms]: min 140004.9 avg 140005.7 max 140006.4 median 0.0 stddev
0.6
Connection time [ms]: connect 0.8
Connection length [replies/conn]: 15.000
Request rate: 0.1 req/s (9333.7 ms/req)
Request size [B]: 67.0
Reply rate [replies/s]: min 0.0 avg 0.1 max 0.6 stddev 0.1 (112 samples)
Reply time [ms]: response 0.7 transfer 0.0
Reply size [B]: header 283.0 content 177.0 footer 0.0 (total 460.0)
Reply status: 1xx=0 2xx=60 3xx=0 4xx=0 5xx=0
CPU time [s]: user 143.09 system 410.65 (user 25.6% system 73.3% total 98.9%)
Net I/O: 0.1 KB/s (0.0*10^6 bps)
Errors: total 0 client-timo 0 socket-timo 0 connrefused 0 connreset 0
Errors: fd-unavail 0 addrunavail 0 ftab-full 0 other 0
Session rate [sess/s]: min 0.00 avg 0.01 max 0.20 stddev 0.03 (4/4)
Session: avg 1.00 connections/session
Session lifetime [s]: 140.0
Session failtime [s]: 0.0
Session length histogram: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 4
```

- **MV 3:**

```
httpperf --hog --client=0/1 --server=192.168.122.53 --port=80 --uri=/ --send-
buffer=4096 --recv-buffer=16384 --wsess=4$
Maximum connect burst length: 1
Total: connections 4 requests 60 replies 60 test-duration 560.061 s
Connection rate: 0.0 conn/s (140015.2 ms/conn, <=2 concurrent connections)
Connection time [ms]: min 140010.8 avg 140015.2 max 140023.2 median 0.0 stddev
5.8
Connection time [ms]: connect 0.6
Connection length [replies/conn]: 15.000
Request rate: 0.1 req/s (9334.3 ms/req)
Request size [B]: 67.0
Reply rate [replies/s]: min 0.0 avg 0.1 max 0.6 stddev 0.1 (112 samples)
Reply time [ms]: response 1.2 transfer 0.0
Reply size [B]: header 283.0 content 177.0 footer 0.0 (total 460.0)
Reply status: 1xx=0 2xx=60 3xx=0 4xx=0 5xx=0
CPU time [s]: user 145.77 system 409.06 (user 26.0% system 73.0% total 99.1%)
Net I/O: 0.1 KB/s (0.0*10^6 bps)
Errors: total 0 client-timo 0 socket-timo 0 connrefused 0 connreset 0
Errors: fd-unavail 0 addrunavail 0 ftab-full 0 other 0
Session rate [sess/s]: min 0.00 avg 0.01 max 0.20 stddev 0.03 (4/4)
Session: avg 1.00 connections/session
Session lifetime [s]: 140.0
Session failtime [s]: 0.0
```

Session length histogram: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 4

- SSD + SSD swap

- **Datos potenciómetro:**

```
$ sudo /tmp/powermeter/powermeter.sh totalenergy
99856.770100
```

```
$ sudo /tmp/powermeter/powermeter.sh totaltime
574.8730
```

- **MV 0:**

```
httpperf --hog --client=0/1 --server=192.168.122.186 --port=80 --uri=/ --send-
buffer=4096 --recv-buffer=16384 --wsess=$
```

Maximum connect burst length: 1

Total: connections 4 requests 60 replies 60 test-duration 560.090 s

Connection rate: 0.0 conn/s (140022.4 ms/conn, <=2 concurrent connections)

Connection time [ms]: min 140009.6 avg 140022.4 max 140050.8 median 0.0 stddev 19.1

Connection time [ms]: connect 0.5

Connection length [replies/conn]: 15.000

Request rate: 0.1 req/s (9334.8 ms/req)

Request size [B]: 68.0

Reply rate [replies/s]: min 0.0 avg 0.1 max 0.6 stddev 0.1 (112 samples)

Reply time [ms]: response 1.7 transfer 0.0

Reply size [B]: header 283.0 content 177.0 footer 0.0 (total 460.0)

Reply status: 1xx=0 2xx=60 3xx=0 4xx=0 5xx=0

CPU time [s]: user 142.77 system 411.95 (user 25.5% system 73.6% total 99.0%)

*Net I/O: 0.1 KB/s (0.0*10⁶ bps)*

Errors: total 0 client-timo 0 socket-timo 0 connrefused 0 connreset 0

Errors: fd-unavail 0 addrunavail 0 ftab-full 0 other 0

Session rate [sess/s]: min 0.00 avg 0.01 max 0.20 stddev 0.03 (4/4)

Session: avg 1.00 connections/session

Session lifetime [s]: 140.0

Session failtime [s]: 0.0

Session length histogram: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 4

- **MV 1:**

```
httpperf --hog --client=0/1 --server=192.168.122.244 --port=80 --uri=/ --send-
buffer=4096 --recv-buffer=16384 --wsess=$
```

Maximum connect burst length: 1

Total: connections 4 requests 60 replies 60 test-duration 560.090 s

Connection rate: 0.0 conn/s (140022.4 ms/conn, <=2 concurrent connections)

Connection time [ms]: min 140010.2 avg 140022.4 max 140050.0 median 0.0 stddev 18.6

Connection time [ms]: connect 2.9

Raúl López Martín

Connection length [replies/conn]: 15.000
Request rate: 0.1 req/s (9334.8 ms/req)
Request size [B]: 68.0
Reply rate [replies/s]: min 0.0 avg 0.1 max 0.6 stddev 0.1 (112 samples)
Reply time [ms]: response 1.6 transfer 0.0
Reply size [B]: header 283.0 content 177.0 footer 0.0 (total 460.0)
Reply status: 1xx=0 2xx=60 3xx=0 4xx=0 5xx=0
CPU time [s]: user 146.63 system 409.46 (user 26.2% system 73.1% total 99.3%)
Net I/O: 0.1 KB/s (0.0*10⁶ bps)
Errors: total 0 client-timo 0 socket-timo 0 connrefused 0 connreset 0
Errors: fd-unavail 0 addrunavail 0 ftab-full 0 other 0
Session rate [sess/s]: min 0.00 avg 0.01 max 0.20 stddev 0.03 (4/4)
Session: avg 1.00 connections/session
Session lifetime [s]: 140.0
Session failtime [s]: 0.0
Session length histogram: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 4

- **MV 2:**

`httpperf --hog --client=0/1 --server=192.168.122.41 --port=80 --uri=/ --send-buffer=4096 --recv-buffer=16384 --wsess=4$`
Maximum connect burst length: 1
Total: connections 4 requests 60 replies 60 test-duration 560.071 s
Connection rate: 0.0 conn/s (140017.7 ms/conn, <=2 concurrent connections)
Connection time [ms]: min 140007.4 avg 140017.7 max 140037.0 median 0.0 stddev 13.2
Connection time [ms]: connect 0.7
Connection length [replies/conn]: 15.000
Request rate: 0.1 req/s (9334.5 ms/req)
Request size [B]: 67.0
Reply rate [replies/s]: min 0.0 avg 0.1 max 0.6 stddev 0.1 (112 samples)
Reply time [ms]: response 1.4 transfer 0.0
Reply size [B]: header 283.0 content 177.0 footer 0.0 (total 460.0)
Reply status: 1xx=0 2xx=60 3xx=0 4xx=0 5xx=0
CPU time [s]: user 145.32 system 412.26 (user 25.9% system 73.6% total 99.6%)
Net I/O: 0.1 KB/s (0.0*10⁶ bps)
Errors: total 0 client-timo 0 socket-timo 0 connrefused 0 connreset 0
Errors: fd-unavail 0 addrunavail 0 ftab-full 0 other 0
Session rate [sess/s]: min 0.00 avg 0.01 max 0.20 stddev 0.03 (4/4)
Session: avg 1.00 connections/session
Session lifetime [s]: 140.0
Session failtime [s]: 0.0
Session length histogram: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 4

- **MV 3:**

`httpperf --hog --client=0/1 --server=192.168.122.68 --port=80 --uri=/ --send-buffer=4096 --recv-buffer=16384 --wsess=4$`
Maximum connect burst length: 1
Total: connections 4 requests 60 replies 60 test-duration 560.045 s

Raúl López Martín

Connection rate: 0.0 conn/s (140011.2 ms/conn, <=2 concurrent connections)
Connection time [ms]: min 140008.6 avg 140011.2 max 140016.3 median 0.0 stddev 3.6
Connection time [ms]: connect 1.2
Connection length [replies/conn]: 15.000
Request rate: 0.1 req/s (9334.1 ms/req)
Request size [B]: 67.0
Reply rate [replies/s]: min 0.0 avg 0.1 max 0.6 stddev 0.1 (112 samples)
Reply time [ms]: response 1.1 transfer 0.0
Reply size [B]: header 283.0 content 177.0 footer 0.0 (total 460.0)
Reply status: 1xx=0 2xx=60 3xx=0 4xx=0 5xx=0
CPU time [s]: user 145.52 system 409.31 (user 26.0% system 73.1% total 99.1%)
Net I/O: 0.1 KB/s (0.0*10⁶ bps)
Errors: total 0 client-timo 0 socket-timo 0 connrefused 0 connreset 0
Errors: fd-unavail 0 addrunavail 0 ftab-full 0 other 0
Session rate [sess/s]: min 0.00 avg 0.01 max 0.20 stddev 0.03 (4/4)
Session: avg 1.00 connections/session
Session lifetime [s]: 140.0
Session failtime [s]: 0.0
Session length histogram: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 4

- Iodrive + swap SSD

- Datos potenciómetro

```
$ sudo /tmp/powermeter/./powermeter.sh totalenergy  
101167.458300
```

```
$ sudo /tmp/powermeter/./powermeter.sh totaltime  
585.3300
```

- MV0

```
httperf --hog --client=0/1 --server=192.168.122.39 --port=80 --uri=/ --send-  
buffer=4096 --recv-buffer=16384 --wsess=4,15,10.000  
Maximum connect burst length: 1  
Total: connections 4 requests 60 replies 60 test-duration 560.030 s  
Connection rate: 0.0 conn/s (140007.5 ms/conn, <=2 concurrent connections)  
Connection time [ms]: min 140005.8 avg 140007.5 max 140009.4 median 0.0 stddev 1.9  
Connection time [ms]: connect 0.7  
Connection length [replies/conn]: 15.000  
Request rate: 0.1 req/s (9333.8 ms/req)  
Request size [B]: 67.0  
Reply rate [replies/s]: min 0.0 avg 0.1 max 0.6 stddev 0.1 (112 samples)  
Reply time [ms]: response 0.8 transfer 0.0  
Reply size [B]: header 283.0 content 177.0 footer 0.0 (total 460.0)  
Reply status: 1xx=0 2xx=60 3xx=0 4xx=0 5xx=0
```

Raúl López Martín

CPU time [s]: user 144.67 system 409.05 (user 25.8% system 73.0% total 98.9%)
Net I/O: 0.1 KB/s (0.0*10⁶ bps)
Errors: total 0 client-timo 0 socket-timo 0 connrefused 0 connreset 0
Errors: fd-unavail 0 addrunavail 0 ftab-full 0 other 0
Session rate [sess/s]: min 0.00 avg 0.01 max 0.20 stddev 0.03 (4/4)
Session: avg 1.00 connections/session
Session lifetime [s]: 140.0
Session failtime [s]: 0.0
Session length histogram: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 4

- **MV1**

httpperf --hog --client=0/1 --server=192.168.122.58 --port=80 --uri=/ --send-buffer=4096 --recv-buffer=16384 --wsess=4,15,10.000
Maximum connect burst length: 1
Total: connections 4 requests 60 replies 60 test-duration 560.073 s
Connection rate: 0.0 conn/s (140018.2 ms/conn, <=2 concurrent connections)
Connection time [ms]: min 140010.6 avg 140018.2 max 140029.0 median 0.0 stddev 8.1
Connection time [ms]: connect 0.6
Connection length [replies/conn]: 15.000
Request rate: 0.1 req/s (9334.5 ms/req)
Request size [B]: 67.0
Reply rate [replies/s]: min 0.0 avg 0.1 max 0.6 stddev 0.1 (112 samples)
Reply time [ms]: response 1.4 transfer 0.0
Reply size [B]: header 283.0 content 177.0 footer 0.0 (total 460.0)
Reply status: 1xx=0 2xx=60 3xx=0 4xx=0 5xx=0
CPU time [s]: user 145.15 system 409.78 (user 25.9% system 73.2% total 99.1%)
Net I/O: 0.1 KB/s (0.0*10⁶ bps)
Errors: total 0 client-timo 0 socket-timo 0 connrefused 0 connreset 0
Errors: fd-unavail 0 addrunavail 0 ftab-full 0 other 0
Session rate [sess/s]: min 0.00 avg 0.01 max 0.20 stddev 0.03 (4/4)
Session: avg 1.00 connections/session
Session lifetime [s]: 140.0
Session failtime [s]: 0.0
Session length histogram: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 4

- **MV2**

httpperf --hog --client=0/1 --server=192.168.122.117 --port=80 --uri=/ --send-buffer=4096 --recv-buffer=16384 --wsess=4,15,10.000
Maximum connect burst length: 1
Total: connections 4 requests 60 replies 60 test-duration 560.075 s
Connection rate: 0.0 conn/s (140018.8 ms/conn, <=2 concurrent connections)
Connection time [ms]: min 140005.2 avg 140018.8 max 140031.3 median 0.0 stddev 11.0
Connection time [ms]: connect 1.8
Connection length [replies/conn]: 15.000
Request rate: 0.1 req/s (9334.6 ms/req)
Request size [B]: 68.0

Raúl López Martín

Reply rate [replies/s]: min 0.0 avg 0.1 max 0.6 stddev 0.1 (112 samples)
Reply time [ms]: response 1.3 transfer 0.0
Reply size [B]: header 283.0 content 177.0 footer 0.0 (total 460.0)
Reply status: 1xx=0 2xx=60 3xx=0 4xx=0 5xx=0
CPU time [s]: user 145.46 system 409.11 (user 26.0% system 73.0% total 99.0%)
Net I/O: 0.1 KB/s (0.0*10⁶ bps)
Errors: total 0 client-timo 0 socket-timo 0 connrefused 0 connreset 0
Errors: fd-unavail 0 addrunavail 0 ftab-full 0 other 0
Session rate [sess/s]: min 0.00 avg 0.01 max 0.20 stddev 0.03 (4/4)
Session: avg 1.00 connections/session
Session lifetime [s]: 140.0
Session failtime [s]: 0.0
Session length histogram: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 4

- **MV3**

httpperf --hog --client=0/1 --server=192.168.122.159 --port=80 --uri=/ --send-buffer=4096 --recv-buffer=16384 --wsess=4,15,10.000
Maximum connect burst length: 1
Total: connections 4 requests 60 replies 60 test-duration 560.066 s
Connection rate: 0.0 conn/s (140016.4 ms/conn, <=2 concurrent connections)
Connection time [ms]: min 140011.7 avg 140016.4 max 140026.0 median 0.0 stddev 6.7
Connection time [ms]: connect 3.0
Connection length [replies/conn]: 15.000
Request rate: 0.1 req/s (9334.4 ms/req)
Request size [B]: 68.0
Reply rate [replies/s]: min 0.0 avg 0.1 max 0.6 stddev 0.1 (112 samples)
Reply time [ms]: response 1.2 transfer 0.0
Reply size [B]: header 283.0 content 177.0 footer 0.0 (total 460.0)
Reply status: 1xx=0 2xx=60 3xx=0 4xx=0 5xx=0
CPU time [s]: user 145.95 system 410.46 (user 26.1% system 73.3% total 99.3%)
Net I/O: 0.1 KB/s (0.0*10⁶ bps)
Errors: total 0 client-timo 0 socket-timo 0 connrefused 0 connreset 0
Errors: fd-unavail 0 addrunavail 0 ftab-full 0 other 0
Session rate [sess/s]: min 0.00 avg 0.01 max 0.20 stddev 0.03 (4/4)
Session: avg 1.00 connections/session
Session lifetime [s]: 140.0
Session failtime [s]: 0.0
Session length histogram: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 4

b) Sysbench

- HDD + swap HDD

- Datos potenciómetro

```
$ sudo /tmp/powermeter/powermeter.sh totalenergy  
47728.673400
```

```
$ sudo /tmp/powermeter/./powermeter.sh totaltime  
269.2310
```

- Resultados

OLTP test statistics:

queries performed:

<i>read:</i>	1186990
<i>write:</i>	121389
<i>other:</i>	20145
<i>total:</i>	1328524

<i>transactions:</i>	1000 (3.79 per sec.)
<i>deadlocks:</i>	18145 (68.78 per sec.)
<i>read/write requests:</i>	1308379 (4959.62 per sec.)
<i>other operations:</i>	20145 (76.36 per sec.)

Test execution summary:

<i>total time:</i>	263.8062s
<i>total number of events:</i>	1000
<i>total time taken by event execution:</i>	98664.8087

per-request statistics:

<i>min:</i>	2777.06ms
<i>avg:</i>	98664.81ms
<i>max:</i>	263749.92ms
<i>approx. 95 percentile:</i>	252664.38ms

Threads fairness:

<i>events (avg/stddev):</i>	2.5000/1.20
<i>execution time (avg/stddev):</i>	246.6620/14.78

- HDD + swap SSD

- Datos potenciómetro

```
$ sudo /tmp/powermeter/./powermeter.sh totalenergy  
55015.292400
```

```
$ sudo /tmp/powermeter/./powermeter.sh totaltime  
351.7740
```

- Resultados

OLTP test statistics:

queries performed:

read: 1144024

write: 117721

other: 19452

total: 1281197

transactions: 1000 (3.97 per sec.)

deadlocks: 17452 (69.30 per sec.)

read/write requests: 1261745 (5010.39 per sec.)

other operations: 19452 (77.24 per sec.)

Test execution summary:

total time: 251.8258s

total number of events: 1000

total time taken by event execution: 94710.6396

per-request statistics:

min: 2298.86ms

avg: 94710.64ms

max: 251618.12ms

approx. 95 percentile: 243021.03ms

Threads fairness:

events (avg/stddev): 2.5000/1.26

execution time (avg/stddev): 236.7766/13.79

- SSD + swap SSD

- Datos potenciómetro

```
$ sudo /tmp/powermeter/./powermeter.sh totalenergy  
45105.837000
```

```
$ sudo /tmp/powermeter/./powermeter.sh totaltime  
261.4800
```

- Resultados

```
OLTP test statistics:
queries performed:
  read:          1197778
  write:         121533
  other:         20319
  total:         1339630
transactions:    1000 (4.05 per sec.)
deadlocks:      18319 (74.26 per sec.)
read/write requests: 1319311 (5348.16 per sec.)
other operations: 20319 (82.37 per sec.)
Test execution summary:
total time:      246.6852s
total number of events: 1000
total time taken by event execution: 92174.8071
per-request statistics:
  min:           2676.99ms
  avg:           92174.81ms
  max:           246590.46ms
  approx. 95 percentile: 238053.06ms

Threads fairness:
events (avg/stddev): 2.5000/1.33
execution time (avg/stddev): 230.4370/14.29
```

- Iodrive + swap SSD

- Datos potenciómetro

```
$ sudo /tmp/powermeter/./powermeter.sh totalenergy
45035.5118
```

```
$ sudo /tmp/powermeter/./powermeter.sh totaltime
265.2900
```

- Resultados

```
OLTP test statistics:
queries performed:
  read:          1228096
  write:         121853
  other:         20808
  total:         1370757
transactions:    1000 (3.99 per sec.)
deadlocks:      18808 (75.11 per sec.)
read/write requests: 1349949 (5390.69 per sec.)
other operations: 20808 (83.09 per sec.)
```

Test execution summary:

total time: 250.4223s
total number of events: 1000
total time taken by event execution: 92862.4276
per-request statistics:
min: 2772.07ms
avg: 92862.43ms
max: 250329.77ms
approx. 95 percentile: 239554.20ms

Threads fairness:

events (avg/stddev): 2.5000/1.28
execution time (avg/stddev): 222.1561/14.52

c) **b_eff_io**

- HDD + swap HDD

- Datos potenciómetro

```
$ sudo /tmp/powermeter/./powermeter.sh totalenergy  
102303,6841
```

```
$ sudo /tmp/powermeter/./powermeter.sh totaltime  
603,593
```

- Resultado

```
MEMORY_PER_PROCESSOR = 1024 MBytes [1MBytes = 1024*1024 bytes,  
1MB = 1e6 bytes]
```

```
Maximum chunk size = 8.000 MBytes
```

```
-N 2,3,4 T=600, MT=4096 MBytes, -D WITHOUT_SHARED, -noshared  
PATH=/mirror/scratch, PREFIX=/mirror/b_eff_io/b_eff_io_HDD1_N4
```

```
system name : Linux
```

```
hostname : vmOMPINode0
```

```
OS release : 2.6.32-5-686
```

```
OS version : #1 SMP Sun Sep 23 09:49:36 UTC 2012
```

```
machine : i686
```

```
Date of measurement: Thu Jan 17 22:56:11 2013
```

```
weighted average bandwidth for write : 1.497 MB/s on 2 processes  
weighted average bandwidth for rewrite : 3.956 MB/s on 2 processes  
weighted average bandwidth for read : 7.841 MB/s on 2 processes  
b_eff_io of these measurements = 4.057 MB/s on 2 processes with 1024  
MByte/PE and scheduled time=3.3 min
```

```
weighted average bandwidth for write : 1.565 MB/s on 3 processes  
weighted average bandwidth for rewrite : 4.593 MB/s on 3 processes
```

Raúl López Martín

*weighted average bandwidth for read : 7.217 MB/s on 3 processes
b_eff_io of these measurements = 4.080 MB/s on 3 processes with 1024
MByte/PE and scheduled time=3.3 min*

*weighted average bandwidth for write : 1.646 MB/s on 4 processes
weighted average bandwidth for rewrite : 3.465 MB/s on 4 processes
weighted average bandwidth for read : 7.288 MB/s on 4 processes*

*b_eff_io of these measurements = 3.994 MB/s on 4 processes with 1024
MByte/PE and scheduled time=3.3 min*

- HDD + swap SSD

- Datos potenciómetro

```
$ sudo /tmp/powermeter/./powermeter.sh totalenergy  
100381,7235
```

```
$ sudo /tmp/powermeter/./powermeter.sh totaltime  
609,319
```

- Resultado

```
MEMORY_PER_PROCESSOR = 1024 MBytes [1MBytes = 1024*1024 bytes,  
1MB = 1e6 bytes]
```

```
Maximum chunk size = 8.000 MBytes
```

```
-N 2,3,4 T=600, MT=4096 MBytes, -D WITHOUT_SHARED, -noshared  
PATH=/mirror/scratch, PREFIX=/mirror/b_eff_io/b_eff_io_HDD2_N4
```

```
system name : Linux
```

```
hostname : vmOMPINode0
```

```
OS release : 2.6.32-5-686
```

```
OS version : #1 SMP Sun Sep 23 09:49:36 UTC 2012
```

```
machine : i686
```

```
Date of measurement: Thu Jan 17 23:50:22 2013
```

*weighted average bandwidth for write : 1.394 MB/s on 2 processes
weighted average bandwidth for rewrite : 3.721 MB/s on 2 processes
weighted average bandwidth for read : 7.755 MB/s on 2 processes
b_eff_io of these measurements = 3.951 MB/s on 2 processes with 1024
MByte/PE and scheduled time=3.3 min*

*weighted average bandwidth for write : 2.442 MB/s on 3 processes
weighted average bandwidth for rewrite : 4.395 MB/s on 3 processes
weighted average bandwidth for read : 8.996 MB/s on 3 processes
b_eff_io of these measurements = 5.057 MB/s on 3 processes with 1024
MByte/PE and scheduled time=3.3 min*

Raúl López Martín

weighted average bandwidth for write : 2.692 MB/s on 4 processes
weighted average bandwidth for rewrite : 3.650 MB/s on 4 processes
weighted average bandwidth for read : 8.274 MB/s on 4 processes
b_eff_io of these measurements = 4.978 MB/s on 4 processes with 1024 MByte/PE and scheduled time=3.3 min

- SSD + swap SSD

- Datos potenciómetro

```
$ sudo /tmp/powermeter/./powermeter.sh totalenergy  
98367,7912
```

```
$ sudo /tmp/powermeter/./powermeter.sh totaltime  
605,787
```

- Resultado

```
MEMORY_PER_PROCESSOR = 1024 MBytes [1MBytes = 1024*1024 bytes,  
1MB = 1e6 bytes]
```

```
Maximum chunk size = 8.000 MBytes
```

```
-N 1,2 T=200, MT=2048 MBytes, -D WITHOUT_SHARED, -noshared
```

```
PATH=/mirror/scratch, PREFIX=/mirror/b_eff_io/b_eff_io_last_1
```

```
system name : Linux
```

```
hostname : vmOMPINode0
```

```
OS release : 2.6.32-5-686
```

```
OS version : #1 SMP Sun Sep 23 09:49:36 UTC 2012
```

```
machine : i686
```

```
Date of measurement: Mon Jan 14 19:39:50 2013
```

weighted average bandwidth for write : 4.361 MB/s on 2 processes
weighted average bandwidth for rewrite : 10.634 MB/s on 2 processes
weighted average bandwidth for read : 6.238 MB/s on 2 processes
b_eff_io of these measurements = 5.854 MB/s on 2 processes with 1024 MByte/PE
and scheduled time=1.7 min

weighted average bandwidth for write : 5.847 MB/s on 3 processes
weighted average bandwidth for rewrite : 10.099 MB/s on 3 processes
weighted average bandwidth for read : 7.607 MB/s on 3 processes
7.030 MB/s on 3 processes with 1024 MByte/PE and scheduled time=3.3 min

weighted average bandwidth for write : 4.992 MB/s on 4 processes
weighted average bandwidth for rewrite : 8.955 MB/s on 4 processes
weighted average bandwidth for read : 7.140 MB/s on 4 processes
b_eff_io of these measurements = 6.460 MB/s on 4 processes with 1024 MByte/PE
and scheduled time=3.3 min

- Iodrive + swap SSD

- Datos potenciómetro

```
$ sudo /tmp/powermeter/./powermeter.sh totalenergy  
90967.3273
```

```
$ sudo /tmp/powermeter/./powermeter.sh totaltime  
610.7500
```

- Resultado

MEMORY_PER_PROCESSOR = 1024 MBytes [1MBytes = 1024*1024 bytes,
1MB = 1e6 bytes]

Maximum chunk size = 8.000 MBytes

-N 1,2,3 T=200, MT=3072 MBytes, -D WITHOUT_SHARED, -noshared
PATH=/mirror/scratch, PREFIX=/mirror/b_eff_io/b_eff_io_iodrive3

system name : Linux

hostname : vmOMPINode0

OS release : 2.6.32-5-686

OS version : #1 SMP Sun Sep 23 09:49:36 UTC 2012

machine : i686

Date of measurement: Thu Jan 17 16:06:28 2013

weighted average bandwidth for write : 4.099 MB/s on 2 processes
weighted average bandwidth for rewrite : 9.940 MB/s on 2 processes
weighted average bandwidth for read : 6.745 MB/s on 2 processes
b_eff_io of these measurements = 5.974 MB/s on 2 processes with 1024
MByte/PE and scheduled time=3.3 min

weighted average bandwidth for write : 5.903 MB/s on 3 processes
weighted average bandwidth for rewrite : 10.771 MB/s on 3 processes
weighted average bandwidth for read : 7.704 MB/s on 3 processes
b_eff_io of these measurements = 7.137 MB/s on 3 processes with 1024
MByte/PE and scheduled time=3.3 min

weighted average bandwidth for write : 5.859 MB/s on 4 processes
weighted average bandwidth for rewrite : 9.350 MB/s on 4 processes
weighted average bandwidth for read : 7.260 MB/s on 4 processes

b_eff_io of these measurements = 6.834 MB/s on 4 processes with 1024
MByte/PE and scheduled time=3.3 min

7. Referencias

- [1] **MPI: A Message-Passing Interface Standard**, November 15, 2003, University of Tennessee
- [2] Snir, Marc; Otto, Steve; Huss-Lederman, Steven; Walker, David; Dongarra, Jack; **MPI-The Complete Reference**, 1995, MIT Press Cambridge
- [3] LibNBC <http://www.unixer.de/research/nbcoll/libnbc/>
- [4] Chen, Gari; Gillen, Al; **KVM for server virtualization : An Open Source solution comes of age**; October 2011
- [5] **KVM Kernel based virtualization driver white paper**; Qumranet; 2006
- [6] Freescale <http://www.freescale.com/infocenter/index.jsp?topic=%2FQORIQSDK%2F2412450.html>
- [7] Sysbench project source forge <http://sysbench.sourceforge.net/docs/>
- [8] Man bf_eff_io https://fs.hls.de/projects/par/mpi/b_eff_io/b_eff_io_man.html
- [9] Man httpperf <http://www.hpl.hp.com/research/linux/httpperf/httpperf-man-0.9.txt>
- [10] Mosberger, D; Jin; **HTTPERF: A tool for measuring web server performance**; Performance Evaluation Review, Volume 26, Number 3; December 1998 <http://www.hpl.hp.com/research/linux/httpperf/wisp98/httpperf.pdf>
- [11] Wikipedia **HDD** http://es.wikipedia.org/wiki/Disco_duro
- [12] Wikipedia Inglesa **HDD Magnetic recording** http://en.wikipedia.org/wiki/Hard_disk_drive#Magnetic_recording
- [13] **Constellation.2 HDD Specs** <http://www.seagate.com/files/www-content/product-content/constellation-fam/constellation/constellation-2/en-us/docs/constellation2-fips-ds1719-4-1207us.pdf>
- [14] Wikipedia Inglesa **SDD** http://en.wikipedia.org/wiki/Solid-state_drive
- [15] Ekker, Neal; Coughlin, Tom; Handy, Jim; **An introduction to Solid State Storage**; Texas Memory Systems, Coughlin Associates, Objective_Analysis; January 2009

- [16] White paper: **SSD Power Savings Render Significant Reduction to TCO;** STEC
- [17] **The fusion-io io difference ssd differentiator** <http://www.fusionio.com/white-papers/the-fusion-io-difference-ssd-differentiator/>
- [18] **Taming the power hungry datacenters** <http://www.fusionio.com/white-papers/taming-the-power-hungry-data-center/>
- [19] **ioDrive2 Duo DataSheet** <http://www.fusionio.com/data-sheets/iodrive2-duo/>
- [20] **Watts up** <https://www.wattsupmeters.com/secure/products.php?pn=0&wai=512&spec=3>
- [21] <http://www.howtoforge.com/virtualization-with-kvm-on-a-fedora-17-server>
- [22] IBM <http://publib.boulder.ibm.com/infocenter/lnxinfo/v3r0m0/index.jsp?topic=/liai/kvminstall/liaaikvminstallstart.htm>
- [23] **Documentación de postgresQL 8.4** <http://www.postgresql.org/docs/8.4/static/>
- [24] <http://www.awven.com/q143-mpich2-como-montar-un-cluster-linux-ubuntu/>
- [25] <http://forums.debian.net/viewtopic.php?f=16&t=71654>
- [26] **MPICH2 Installer's Guide**, Pavan, Balaji; Darius, Buntinas; Ralph, Butler; Anthony, Chan; David, Goodell; William, Gropp; Jayesh, Krishna; Rob, Latham; Ewing, Lusk; Guillaume, Mercier; Rob, Ross; Rajeev, Thakur, <http://www.mpich.org/static/docs/guides/mpich2-1.5-installguide.pdf>
- [27] **[PATCH 00/21] PCI: use busn_res to replace bus secondary/subordinate** <http://comments.gmane.org/gmane.linux.kernel/1292971>
- [28] **pci.h** <http://www.cs.fsu.edu/~baker/devices/lxr/http/source/linux/include/linux/pci.h>
- [29] **sysbench wiki** <http://www.percona.com/docs/wiki/benchmark:sysbench:olpt.lua>
- [30] **ERROR 2003 (HY000)** <http://frlx.wordpress.com/2010/12/12/error-2003-hy000-cant-connect-to-mysql-server-on/>
- [31] **sysbench-linux-test-bench** <http://beerpla.net/2007/10/12/sysbench-linux-test-bench/>

- [32] **mysql debian-6-squeeze** <http://library.linode.com/databases/mysql/debian-6-squeeze>
- [33] UNICAN <https://www.meteo.unican.es/trac/meteo/blog/MonitorizacionNFS>