Universidad Carlos III de Madrid
Escuela Politécnica Superior

Ingeniería en Informática
Proyecto Fin de Carrera

Migración y evaluación de un sistema de E/S paralela para plataformas Windows

Autor: Sergio García Linares
Tutores: Francisco Javier García Blas
Florin Daniel Isalia

Julio, 2009

Agradecimientos:

En primer lugar quiero agradecer a mi tutor Javier, por haber estado siempre que le he necesitado y por toda la ayuda e interés que ha mostrado, ya que sin él no hubiera sido posible la realización de este proyecto.

Agradecer también a Patricia, por estar ahí siempre que lo he necesitado y aguantarme en esos malos momentos que se viven durante la carrera.

A mis padres por apoyarme en todo momento en la realización de mis metas, y enseñarme tantas cosas que no se aprenden en los libros.

A mis compañeros y amigos de clase, por esos buenos momentos pasados de risas, reuniones y demás, que han hecho que la carrera sea más llevadera, gracias a vosotros se me quedan más de un grato recuerdo de toda esta vivencia que ha sido la carrera.

Por último y no menos importante, a todos mis amigos por su apoyo y comprensión durante toda la carrera.

A todos los que me he dejado sin mencionar, que sepan que también les tengo muy presentes y les agradezco todo su apoyo y felicidad mostrada.

ÍNDICE DE CONTENIDOS

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

1. INTRODUCCIÓN

En este apartado se describen los objetivos del Proyecto Fin de Carrera, aquellos términos y definiciones del ámbito del proyecto y la estructura que seguirá el documento.

1.1. MOTIVACIÓN

En la actualidad cada vez es más necesaria la ejecución paralela de aplicaciones con el objetivo de aumentar el rendimiento, es decir, resolver problemas grandes de una manera más rápida y económica. Para poder cumplir con las exigencias de cálculo que supone resolver este tipo de problemas, se necesita abandonar el clásico sistema centralizado de computación por un sistema paralelo. Los sistemas cluster nos permiten que las computadoras que lo integran trabajen cooperativamente y en paralelo como si fuera una sola, aumentando considerablemente la capacidad de cómputo.

Uno de los problemas más importantes a los que se enfrenta la computación paralela es la llamada “la crisis de E/S”. Este problema se origina en el cuello de botella que producen los sistemas de E/S. Este cuello de botella puede afectar considerablemente el tiempo de ejecución de las aplicaciones paralelas.

La motivación del desarrollo para el sistema operativo Windows® bajo la plataforma .NET, viene motivada por la mejoría en la posición de supercomputadores, que ejecutan bajo este sistema operativo, dentro del Top500 de los supercomputadores, viéndose como un posible mercado en auge. Se advierte la fuerza que están obteniendo los clusters basados en Windows® dentro de la lista Top500 (los 500 supercomputadores más potentes del mundo), obteniendo una mejora gradual, clasificando en junio de 2008 un sistema construido en colaboración con el Centro Nacional para Aplicaciones de Supercomputación (NCSA), en el puesto 23º con una puntuación de Linpack de 68,5 teraflops y clasificando en noviembre de 2008 un equipo con el sistema Windows HPC construido por el Shanghai Supercomputer Center alcanzando los 180,6 teraflops en el puesto 10. En la última lista oficial realizada en junio de 2009, el supercomputador desarrollado por el centro de supercomputadoras de Sanghai se sitúa en el puesto 15 de la lista.

En la actualidad, los sistemas de ficheros paralelos existentes para la plataforma Windows®, como pueden ser GPFS y Lustre, ofrecen grandes restricciones y dificultades. GPFS solo es válido para arquitecturas de 64 bits y arroja dificultades con más de 32 máquinas. Lustre es un entorno muy complicado de instalar debido a que se hace necesaria la instalación de drivers específicos a la máquina.

La finalidad de este Proyecto Fin de Carrera será el desarrollo de un sistema que permita de una manera sencilla y con las menores restricciones posibles, explotar al máximo la E/S paralela para los supercomputadores basados en el sistema operativo Windows®, mediante la utilización del estándar MPI y de la interfaz de E/S paralela ROMIO.

1.2. OBJETIVOS

A continuación se exponen mediante una breve descripción los objetivos que se han fijado para la realización del presente proyecto:

- Estudio de los sistemas de ficheros paralelos existentes para las plataformas Windows®. Para ello será necesario indagar sobre las posibilidades, tanto comerciales como de software libre disponibles.
- Explorar las posibilidades de Windows® como sistema HPC (High Performance Computing). Este estudio se centrará en las capacidades de E/S, no en las de computación.
- Estudiar y analizar el sistema de E/S paralela AHPIOS y conocer todas sus posibilidades de funcionamiento.
- Migrar la solución comentada en el punto anterior, desarrollado en Linux, a un entorno Windows®.
- Evaluar el sistema y averiguar cuál es la configuración más recomendable.

1.3. DEFINICIONES Y ACRÓNIMOS

A continuación, se definen las palabras técnicas y la nomenclatura utilizada.

Definiciones

- *Benchmark:* es una técnica utilizada para medir el rendimiento de un sistema o componente de un sistema, frecuentemente en comparación con el cual se refiere específicamente a la acción de ejecutar un *benchmark*.
- *Cluster*: conjuntos o conglomerados de computadoras construidos mediante la utilización de componentes de hardware comunes y que se comportan como si fuesen una única computadora.
- *Middleware*: es un software de conectividad que ofrece un conjunto de servicios que hacen posible el funcionamiento de aplicaciones distribuidas sobre plataformas heterogéneas.
- *Round-robin*: es un método de planificación para seleccionar todos los elementos en un grupo de manera equitativa y en un orden, comenzándose normalmente por el primer elemente de la lista hasta llegar al último y volviendo a empezar desde el primero.

Acrónimos

Acrónimo	Significado
ADIO	*Abstract-Device Interface for E/S.*
ROMIO	Implementación portable de MPI-IO, de altas prestaciones.
MPICH	Implementación portable y gratuita de MPI.
MPI-IO	Interfaz de E/S de ficheros paralelos para MPI.
MPE	*Multi-Processing Environment.*
NFS	*Network File System*. Sistema de archivos de red

RPC	*Remote Procedure Call*. Llamada de procedimientos remotos
API	*Application Program Interface.* Interfaz de aplicación
SO	Sistema Operativo.
GPFS	*General Parallel File System*. Sistema de archivos virtuales paralelos
PVFS	*Parallel Virtual File System*. Sistema de archivos virtuales paralelos
PVM	*Parallel Virtual Machine.* Software de código libre que creó un supercomputador virtual.
SOA	*Service Oriented Architecture.* Arquitectura software de utilización de servicios.
HPCBP	*High Performance Computing Basic Profile.*
OGF	*Open Grid Forum.*

Tabla 1: Definiciones y Acrónimos

1.4. ESTRUCTURA DEL DOCUMENTO

En este apartado se detalla cada uno de los capítulos de los que se compone este documento.

En el capítulo 1, *INTRODUCCIÓN*, se presenta el proyecto, proporcionando un breve resumen y los objetivos del mismo.

En el capítulo 2, *ESTADO DEL ARTE*, se proporciona una visión general de la tecnología que engloba el proyecto.

En el capítulo 3, *.NET*, se explica el funcionamiento y la estructura de esta técnica para desarrollar aplicaciones para el entorno Windows®.

En el capítulo 4, *HERRAMIENTAS DE EVALUACIÓN DE E/S*, se especifican las técnicas utilizadas para realizar los test de rendimiento al sistema.

En el capítulo 5, *SISTEMA DE E/S PARALELA AHPIOS*, se proporciona una visión sobre el funcionamiento, desarrollo e implementación del sistema de ficheros.

En el capítulo 6, *ETAPAS DE LA MIGRACIÓN,* se explica cómo se ha realizado la migración del sistema (pasos, dificultades encontradas, etc).

En el capítulo 7, *EVALUACIÓN*, se mostrará una evaluación descriptiva y gráfica sobre los resultados obtenidos.

En el capítulo 8, *CONCLUSIONES Y TRABAJOS FUTUROS*, se realizará un balance de los objetivos logrados y se expondrán las conclusiones obtenidas de la realización del proyecto.

En el capítulo, *REFERENCIAS*, se expondrá la bibliografía usada en el proyecto.

Finalmente, se incluirá un apéndice I, denominado "MANUAL DE INSTALACIÓN", en el que se especificarán los requisitos mínimos para la instalación y ejecución del proyecto, así como los pasos a seguir para realizar la instalación de forma adecuada, otro apéndice II, denominado "INFRAESTRUCTURA", en el que se explicará como configurar la infraestructura, poniendo un ejemplo de una configuración y por último un apéndice III, denominado "MANUAL DE USUARIO", el que se detalla cómo realizar la compilación y ejecución de un programa paralelo.

1.5. CARACTERÍSTICAS DEL DOCUMENTO

En este punto es necesario entablar las particularidades de este documento. Este documento tiene las siguientes características:

- Los títulos vendrán en mayúsculas, negrita y con letra Times New Roman, con un tamaño de 16, excepto los títulos de primer nivel que tendrán un tamaño de 22.
- El resto de texto vendrá dado en letra Times New Roman con un tamaño de 12 puntos y un interlineado de 1,5.
- Todas las figuras vendrán numeradas y con su correspondiente título descriptivo que será el que aparezca en el índice correspondiente.

2. ESTADO DEL ARTE

Durante este apartado se proporciona una visión general de la tecnología que engloba el proyecto. Para ello se describe de forma detallada las distintas tecnologías disponibles actualmente con el fin de utilizar la información y funcionalidad proporcionada por las diversas tecnologías para realizar un diseño eficiente del proyecto.

En primer lugar se realiza una exposición general de la tecnología cluster, estudiándose posteriormente los sistemas de ficheros distribuidos y más concretamente los sistemas de ficheros paralelos. Al tratarse el tema de paralelismo se analizará el estándar MPI (*Message Passing Interface*) y los instrumentos para analizar las aplicaciones paralelas. En último lugar se estudiará la actualidad existente en sistemas de ficheros y cluster en la plataforma Windows®, teniendo así una visión de las posibilidades del sistema operativo.

2.1. CLUSTERS

El origen del término y uso de este tipo de tecnología es desconocido pero se puede considerar que comenzó entre finales de los años 50 y principios de los años 60.

La base formal del término, en cuanto a Ingeniería Informática se refiere, está relacionado con trabajos en paralelo de cualquier tipo. En 1967, Gene Amdahl publicó lo que ha llegado a ser considerado como el inicio del procesamiento paralelo: la Ley de Amdahl [1]. Esta ley que describe matemáticamente el aceleramiento que se puede esperar paralelizando cualquier serie de tareas al ser aplicadas sobre una arquitectura paralela.

La historia de los primeros grupos de computadoras está fuertemente ligada a la historia de los inicios de las redes. Una de las principales motivaciones para el desarrollo de una red era enlazar los distintos recursos que ofrece un conjunto de computadoras.

Utilizando el concepto de red de conmutación de paquetes, el proyecto ARPANET [2] logró crear en 1969 lo que fue posiblemente la primera red de ordenadores básica basada en un cluster de ordenadores formado por cuatro tipos de centros informáticos. El proyecto ARPANET creció y se convirtió en lo que es ahora Internet, que se puede considerar como "la madre de todos los clusters" (como la unión de casi todos los recursos de cómputo, incluidos los clusters, que pasarían a ser conectados).

También estableció el paradigma de uso de computadoras cluster en el mundo de hoy que consiste en el uso de las redes de conmutación de paquetes para realizar las comunicaciones entre procesadores localizados en los marcos de otro modo desconectado.

La construcción de PC's por los clientes y grupos de investigación se desarrolló a la vez que la organización de las redes y el sistema operativo Unix desde principios de la década de los años 70. Estos desarrollos dieron sus frutos, sacando a la luz proyectos

como TCP/IP [3] y el proyecto de la Xerox PARC formado por protocolos basados en redes de comunicaciones. El núcleo del sistema operativo fue construido en 1971.

Sin embargo, no fue hasta alrededor de 1983, cuando los protocolos y herramientas para el trabajo remoto, comenzaron a facilitar la distribución y uso compartido de archivos y recursos (en gran medida dentro del contexto de BSD Unix [4], e implementados por Sun Microsystems).

El primer producto comercial de tipo cluster fue ARCnet [5], desarrollado en 1977 por Datapoint. No obtuvo un éxito comercial y los cluster no consiguieron tener éxito hasta que en 1984 VAXcluster [6] fue desarrollado con el sistema operativo VAX/VMS. El ARCnet y VAXcluster no sólo son productos que apoyan la computación paralela, sino que además también comparten los sistemas de archivos y dispositivos periféricos. La idea era proporcionar las ventajas del procesamiento paralelo, al tiempo que se mantiene la fiabilidad de los datos y el carácter singular.

Otros dos principales clusters comerciales desarrollados más adelante fueron el Tandem Himalaya [7] (alrededor de 1994 con productos de alta disponibilidad) y el IBM S/390 Parallel Sysplex [8] (también alrededor de 1994, principalmente destinado a empresas).

La historia de los clusters de ordenadores no estaría completa sin señalar el papel fundamental desempeñado por el desarrollo del software de Parallel Virtual Machine (PVM) [9]. Este software de código abierto y basado en comunicaciones TCP/IP permitió la creación de un superordenador virtual, un cluster de computación de alto rendimiento (High Performance Computing). De forma libre los clusters heterogéneos han constituido la cima de este modelo logrando aumentar rápidamente en Flops y superando con creces la disponibilidad incluso de los más caros superordenadores.

En 1995, la invención de la arquitectura de tipo "*beowulf*"[10], una granja de computación diseñado en base a un producto básico de la red con el objetivo específico de "ser un superordenador" capaz de realizar de forma eficiente cálculos paralelos. Esta invención estimuló el desarrollo independiente de la computación Grid como una

entidad, a pesar de que el estilo Grid giraba en torno al del sistema operativo Unix y el Arpanet.

- **Beneficios de la Tecnología Cluster**

Las aplicaciones paralelas escalables requieren: buen rendimiento, comunicaciones que dispongan de gran ancho de banda y baja latencia, redes escalables y acceso rápido a los archivos. Un cluster puede satisfacer estos requisitos usando los recursos que tiene asociados a él.

Los cluster ofrecen las siguientes características a un costo relativamente bajo:

- Alto Rendimiento.
- Alta Disponibilidad.
- Alta Eficiencia.
- Escalabilidad.

La tecnología cluster permite a las organizaciones incrementar su capacidad de procesamiento usando tecnología estándar, tanto en componentes de hardware como de software, que pueden adquirirse a un costo relativamente bajo.

- **Clasificación de los Clusters**

El término cluster tiene diferentes connotaciones para diferentes grupos de personas. Los tipos de clusters, establecidos en base al uso que se dé a los clusters y los servicios que ofrecen, determinan el significado del término para el grupo que lo utiliza. Los clusters pueden clasificarse en base a sus características. Los tres tipos de cluster que se pueden tener son:

- **Alto rendimiento**: Son clusters en los cuales se ejecutan tareas que requieren de gran capacidad computacional, grandes cantidades de memoria, o ambos a la vez. El llevar a cabo estas tareas puede comprometer los recursos del cluster por largos periodos de tiempo.

- **Alta disponibilidad**: Son clusters cuyo objetivo de diseño es el de proveer disponibilidad y confiabilidad. Estos clusters tratan de brindar la máxima disponibilidad de los servicios que ofrecen. La confiabilidad se provee mediante software que detecta fallos y permite recuperarse frente a los mismos, mientras que en hardware se evita tener un único punto de fallos.

- **Alta eficiencia**: Son clusters cuyo objetivo de diseño es el ejecutar la mayor cantidad de tareas en el menor tiempo posible. Existe independencia de datos entre las tareas individuales. El retardo entre los nodos del cluster no es considerado un gran problema.

Los clusters pueden ser clasificados también como Clusters de IT Comerciales (Alta disponibilidad, Alta eficiencia) y Clusters Científicos (Alto rendimiento). A pesar de las discrepancias a nivel de requerimientos de las aplicaciones, muchas de las características de las arquitecturas de hardware y software que están por debajo de las aplicaciones en todos estos clusters, son las mismas. Más aún, un cluster de determinado tipo, puede también presentar características de los otros.

2.1.1. CLUSTER DE TIPO BEOWULF

El cluster encuadrado en la tipología *Beowulf* posee una arquitectura basada en multicomputadoras y suele ser utilizado para computación paralela. Este sistema consta de un nodo maestro y uno o más nodos esclavos conectados a través de una red ethernet u otra topología de red. Esta construido con componentes de hardware comunes en el mercado, similar a cualquier PC, y adaptadores de ethernet y switches estándares. Como no contiene elementos especiales es fácilmente reproducible, y por tanto es sencillo implementarlo con unos mínimos recursos básicos.

Una de las diferencias principales entre un cluster *Beowulf* y un cluster de estaciones de trabajo (COW, *cluster of workstations*) se basa en que la visión que ofrece un cluster *Beowulf* se similar a una sola máquina que muchas estaciones de trabajo. En

la mayoría de los casos, los nodos de cómputo no tienen dispositivos de E/S como monitores o teclados y son accedidos solamente vía remota o por terminal serial.

El nodo maestro controla el cluster y presta servicios de sistemas de archivos a los nodos esclavos. Es también la consola del cluster y la conexión hacia el mundo exterior. Los sistemas grandes *Beowulf* pueden tener más de un nodo maestro, y otros nodos dedicados a diversas tareas específicas, como por ejemplo, consolas o estaciones de supervisión. En la mayoría de los casos los nodos esclavos de un sistema *Beowulf* son estaciones simples. Los nodos son configurados y controlados por el nodo maestro, y hacen solamente lo que éste le indique.

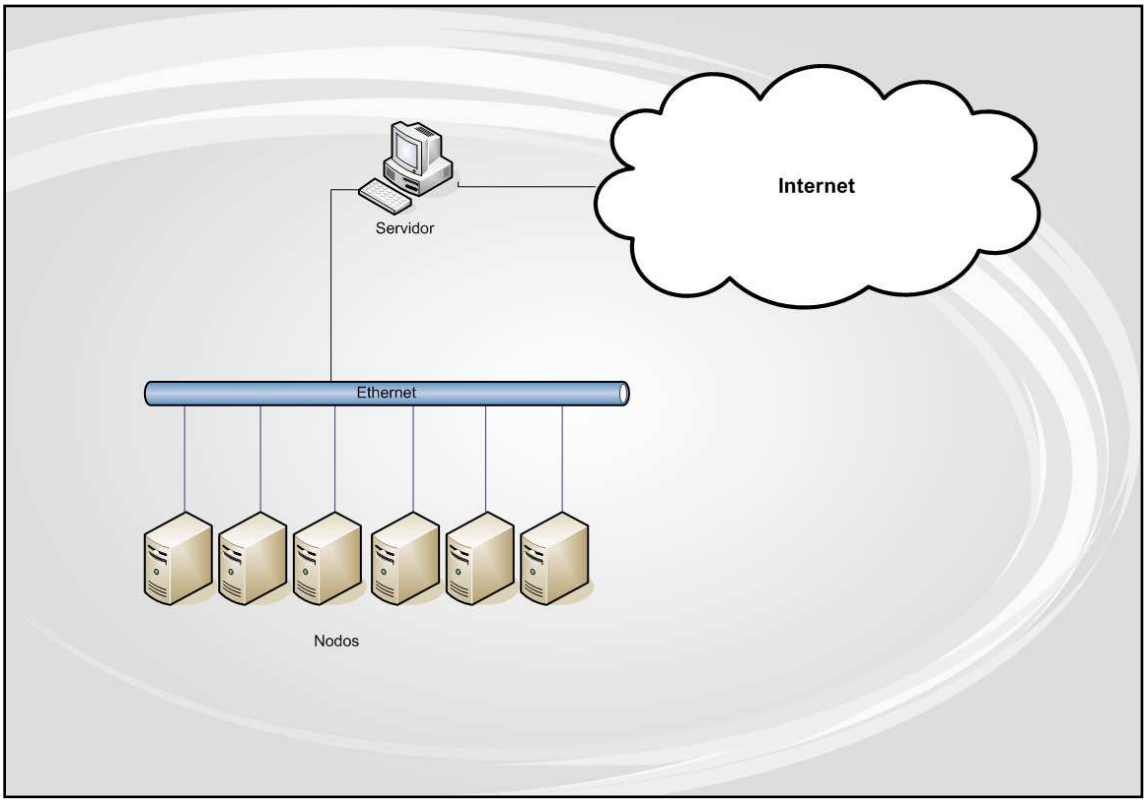


Figura 1: Esquema general de un cluster

2.2. SISTEMAS DE FICHEROS DISTRIBUIDOS

Los sistemas de ficheros distribuidos ofrecen un espacio global de almacenamiento que posibilita a múltiples clientes compartir los datos. En estos sistemas cada fichero se almacena en un servidor, y el ancho de banda de acceso a un fichero se encuentra limitado por el acceso a un único servidor, lo que convierte a los servidores en un cuello de botella en el sistema. Este problema, conocido como la crisis de la E/S, sigue sin estar resuelto en sistemas distribuidos de propósito general, pero se han propuesto diferentes soluciones para resolver este problema, entre las que cabe destacar las siguientes:

- Utilización de paralelismo en el sistema de E/S con la distribución de los datos de un fichero entre diferentes dispositivos y servidores (en este apartado solo trataremos esta solución).
- Empleo de sistemas de almacenamiento de altas prestaciones (redes de almacenamiento).

La idea original de la E/S paralela viene ligada junto a la aparición del sistema RAID (Redundant Array of Inexpensive Disks) [11], a principios de los años 80 por un grupo de investigadores de IC Berkeley, este sistema de almacenamiento hace uso de varios disco duros entre los que distribuye y/o replica los datos, ofreciéndonos con respecto al clásico almacenamiento con un disco duro (dependiendo de la configuración realizada), mayor integridad, tolerancia a fallos, rendimiento y capacidad.

En la actualidad existen principalmente dos alternativas para el empleo de paralelismo en el sistema de E/S:

- Bibliotecas paralelas de E/S.
- Sistemas de ficheros paralelos.

2.2.1. BIBLIOTECAS PARALELAS DE E/S

La computación paralela se utiliza en diferentes tipos de aplicaciones, donde cada una de ellas tiene requisitos muy diferentes. Por este motivo, han surgido diversas bibliotecas de E/S paralela que ofrecen a los programadores de aplicaciones un conjunto de funciones de E/S altamente especializadas y que intentan obtener el máximo rendimiento y flexibilidad para cada clase de aplicación. Algunas de estas bibliotecas son PASSION (*Parallel And Scalable Software for E/S*), Panda, Jovian, DRA (*Disk Resident Arrays*), MIO S, VIP-FS, ChemIO o *VIPIOS*.

El principal problema que reportan las bibliotecas paralelas de E/S es que al estar concebidas fundamentalmente para el desarrollo de aplicaciones paralelas, no ofrecen una solución genérica para su uso en sistemas distribuidos, debido a que cada una de ellas proporciona un API diferente a las aplicaciones, produciendo una falta de estandarización de la E/S paralela.

2.2.1.1. MPI-IO

MPI-IO [17] constituye el único intento real de estandarización de la interfaz paralela de E/S. MPI-IO comenzó como un proyecto de investigación de IBM en 1994, dando soporte a múltiples operaciones de E/S paralelas y optimizaciones, tales como acceso a ficheros no contiguos, E/S colectiva, E/S asíncrona, preasignación de ficheros o punteros compartidos, incluyéndose en 1997 dentro del estándar MPI-2. MPI-IO debe implementarse sobre un determinado sistema de ficheros a fin de garantizar un buen rendimiento en el acceso en paralelo a los ficheros, no obstante, MPI-IO es sólo una especificación, por lo que es necesario utilizar una determinada implementación de la misma. ROMIO es una implementación portable de MPI-IO, realizada en el *Argonne National Laboratory*.

El componente que permite la portabilidad de esta implementación es ADIO (*Abstract Device Interface for Parallel E/S*) [19], que ofrece una interfaz abstracta para E/S paralela. ROMIO es implementado encima de ADIO y sólo ADIO debe ser implementado de forma separada para los diferentes sistemas de ficheros subyacentes.

En la Sección MPI-IO se dan más detalles sobre la arquitectura y funcionamiento de MPI-IO.

2.2.1.2. HDF5

HDF5 (Hierarchical Data Format) es una librería de propósito general y a la vez un formato de ficheros para el almacenamiento de datos científicos. Fue creado para atender las necesidades de científicos e ingenieros trabajando en entornos de computación de altas prestaciones, con un uso intensivo de datos.

HDF5 almacena dos tipos de objetos principales: "datasets" y grupos. Un "dataset" es, esencialmente, una matriz multidimensional de datos, y un grupo es una estructura para organizar los diferentes objetos en un fichero HDF5. Usando estas dos estructuras básicas se puede crear y almacenar casi cualquier tipo de estructura de datos científica, tales como imágenes, vectores, matrices, así como retículas estructuradas y no estructuradas. También se pueden combinar estos elementos en ficheros HDF5 de acuerdo con las necesidades del caso.

Como resultado, HDF5 hace énfasis en la eficiencia del almacenamiento y de la entrada-salida de datos. HDF5-1.8.1, es la última versión oficial disponible.

2.2.2. SISTEMAS DE FICHERO PARALELOS

Un sistema de ficheros paralelo es aquél que elimina el problema del cuello de botella de E/S, agregando de forma lógica múltiples dispositivos de almacenamiento independientes y nodos de E/S mediante un sistema de almacenamiento de alto rendimiento.

En los sistemas de ficheros paralelos el ancho de banda puede incrementarse mediante:

- **Direccionamiento de disco independiente**, mediante el cual el sistema de ficheros puede acceder a datos de diferentes ficheros de forma concurrente.
- **Reparto de los datos en los nodos**, mediante el cual un solo fichero se puede acceder en paralelo.

Un sistema de ficheros paralelo opera, al igual que los sistemas de ficheros distribuidos, independientemente de las aplicaciones ofreciendo una mayor flexibilidad y generalidad que las bibliotecas.

Los sistemas de ficheros paralelos más conocidos son:

- ***GPFS*** (*General Parallel File System*) sistema de ficheros desarrollado por IBM. GPFS es particularmente apropiado en entornos donde los sistemas distribuidos no ofrecen suficiente rendimiento de ancho de banda. Siendo un entorno que permite a los usuarios compartir el acceso a los datos a través de cluster, posibilitando la interacción a través de las interfaces estándar de UNIX.

La fortaleza de GPFS se basa en:

- Mejora el rendimiento del sistema: permite que múltiples nodos accedan simultáneamente a los datos utilizando llamadas estándar del sistema, balanceando la carga y por lo tanto incrementando el ancho de banda disponible por cada nodo.
- Asegura la consistencia de los datos: utiliza un sofisticado sistema de administración que provee la consistencia de los datos mientras permite múltiple e independientes rutas para archivos con el mismo nombre.
- Alta recuperabilidad y disponibilidad de los datos: crea registros "logs" separados para cada uno de los nodos que intervienen en el sistema. Permite administrar el número de replicas que se desea manejar.
- Alta flexibilidad del sistema: los recursos no se encuentran congelados, se puede añadir o quitar discos al sistema mientras este se encuentra montado. Cuando la demanda es muy baja se puede reconfigurar la cargar del sistema a través de todos los discos configurados. También se puede agregar nuevos nodos sin que el sistema sea detenido y puesta en marcha nuevamente.
- Administración simplificada: los comandos de GPFS guardan la configuración en más de un archivo, conocido colectivamente como "cluster de datos". Los comandos de GPFS están diseñados para sincronizar los datos en cada uno de los nodos del sistema. De tal modo se asegura una exacta configuración de los datos.

- ***LUSTRE*** es un sistema de ficheros paralelo diseñado por Cluster File Systems. En Lustre se considera a cada archivo almacenado en el sistema de archivos Lustre un objeto. Lustre presenta a todos los clientes una semántica POSIX estándar y acceso concurrente lectura y escritura para los objetos compartidos. Un sistema de archivos Lustre tiene cuatro unidades funcionales. Estas son: Meta data server (MDS) para almacenar los metadatos; un Object storage target (OST) para guardar los datos reales; un Object storage server (OSS) para manejar los OSTs; cliente(s) para acceder y utilizar los datos. Los OSTs son dispositivos de bloques. Un MDS, OSS, y un OST pueden residir en el mismo nodo o en nodos diferentes. Lustre no administra directamente los OSTs, y delega esta responsabilidad en los OSSs para asegurar la escalabilidad para grandes clusters y supercomputadores.

  En un *Massively Parallell Processor (MPP)*, los procesadores pueden acceder al sistema de archivos Lustre redirigiendo sus peticiones I/O hacia el nodo con el servicio lanzador de tareas si está configurado como un cliente Lustre. Aunque es el método más sencillo, en general proporciona un bajo rendimiento. Una manera ligeramente más complicada de proporcionar un rendimiento global muy bueno consiste en utilizar la biblioteca *liblustre*. Liblustre es una biblioteca de nivel de usuario que permite a los procesadores montar y utilizar el sistema de archivos Lustre como un cliente, sorteando la redirección hacia el nodo de servicio. Utilizando liblustre, los procesadores pueden acceder al sistema de archivos Lustre, incluso si el nodo de servicio en el que se lanzó el trabajo no es un cliente Lustre. Liblustre proporciona un mecanismo para mover datos directamente entre el espacio de aplicación y los OSSs de Lustre sin necesidad de realizar una copia de datos a través del núcleo ligero, logrando así una baja latencia, y gran ancho de banda en el acceso directo de los procesadores al sistema de archivos Lustre.

- ***PVFS*** (*Parallel Virtual File System*) es un sistema de ficheros paralelo, que permite a las aplicaciones, indistintamente de si estas se ejecutan en forma paralela o secuencia, almacenar y acceder a ficheros cuyos datos se encuentran distribuidos a través de un conjunto de servidores de E/S. PVFS ha sido desarrollado para *clusters* Linux, proporcionando un gran ancho de banda en operaciones de lectura y escritura concurrentes realizadas desde múltiples procesos o *threads* a un fichero común. Los datos de los ficheros se distribuyen a través de los discos de las maquinas del clusters Linux y proporcionando la apariencia de un único sistema de ficheros Unix.

  En 2001 arrancó el proyecto PVFS2, en el que se ha reescrito el código y se ha modificado el núcleo de la versión 1 de PVFS para que sea más apropiado para el entorno en el que los sistemas de ficheros paralelos son desplegados. De esta forma se ha conseguido un sistema de ficheros paralelos más adecuado a las necesidades, robusto y de alto rendimiento.

  Las características que posee la versión 2 de PVFS son:

  - Distribución de datos flexible. PVFS1 utiliza el patrón de distribución round-robin para realizar la distribución de datos de los ficheros entre los servidores. PVFS2 no solo permite el uso del patrón de distribución *round-robin* sino que incluye un sistema modular para añadir nuevos patrones de distribución en el sistema y el uso de ello para los nuevos ficheros.
  - Información de metadatos distribuida. PVFS1 contemplaba un único servidor de metadatos, pero se convierte en un único punto de fallos, puesto que los datos no se encuentran replicados, y es un cuello de botella. En PVFS2 los metadatos se encuentran distribuidos en un conjunto de servidores, que pueden coincidir con los servidores de datos.

- Proporciona múltiples interfaces, incluso un interfaz de MPI-IO vía ROMIO y otra para acceder al sistema de ficheros tradicional mediante el uso de un módulo del kernel de Linux.
- Utiliza servidores y clientes sin estado, lo cual, facilita la recuperación de fallos en el sistema.
- Soporte de concurrencia explícito. En PVFS2 los hilos o procesos ligeros son usados cuando son necesarios para proporcionar acceso no bloqueante a todos los dispositivos y evitar la serialización de las operaciones independientes.
- Soporte de redundancia de datos y metadatos. PVFS1 no soporta redundancia de datos. En PVFS2 utiliza el enfoque RAID para proporcionar tolerancia de fallos de disco, pero si un servidor desaparece, los datos que contenía son inaccesibles hasta que el servidor se recupera.
- Soporta clusters heterogéneos.
- Presenta un diseño modular.

En cuanto a los sistemas de ficheros paralelos, su principal problema es que están especialmente pensados para máquinas paralelas y no se integran adecuadamente en entornos distribuidos de propósito general. Además, cada sistema de ficheros paralelo utiliza una estructura de fichero paralelo diferente, incompatible con la de otros sistemas.

2.3. MPI

La Interfaz de Paso de Mensajes (*Message Passing Interface*), es una biblioteca considerada en la actualidad un estándar, que especifica el protocolo de comunicación entre procesos que trabajan en un programa paralelo y forman parte de un sistema paralelo mediante paso de mensajes.

Las características de MPI son:

- Estandarización.
- Rendimiento: optimización de las librerías para cada uno de los sistemas en los que puede ser ejecutado.
- Escalabilidad: gestión de los distintos tipos de tamaño y formato de los mensajes de forma totalmente transparente.
- Portabilidad: existencia de multitud de librerías o implementaciones como MPICH, OPEN-MPI para C, C++ o Fortran, por poner algunos ejemplos.

En la actualidad existen dos versiones principales del estándar: la versión 1.2 (llamada MPI-1), la cual aporta el paso de mensajes y un entorno de ejecución estático y la versión 2.1 (llamada MPI-2), la cual además incluye operaciones de E/S paralela, gestión dinámica de procesos y operaciones de memoria remota (RMA).

La comunicación mediante paso de mensajes puede ser punto a punto o múltiple, para las cuales MPI proporciona una serie de funciones. La comunicación punto a punto, es la establecida entre dos nodos, pudiendo ser síncrona (el remitente se bloquea hasta que el destinatario empieza a recibir el mensaje) o asíncrona (las llamadas no son bloqueantes y realizan peticiones de envió), mientras que la comunicación múltiple es la que interviene varios nodos a la vez.

2.3.1. FUNDAMENTOS DE MPI

Con MPI el número de procesos necesarios se asigna antes de la ejecución del programa, y no se crean procesos adicionales mientras la aplicación se ejecuta. A cada proceso se le asigna una variable que se denomina *rank*, la cual identifica a cada proceso, en el rango de 0 a p-1 donde p es el número total de procesos, realizándose el control de ejecución del programa mediante el uso de esta variable, ya que posibilita controlar que proceso ejecuta una determinada sección de código. Además en MPI se define un comunicador como una colección de procesos, los cuales pueden enviar mensajes entre ellos y permite abstraerse de la topología a nivel físico de las computadoras, el comunicador básico se denomina MPI_COMM_WORLD el cual agrupa a todos los procesos activos durante la ejecución de una aplicación.

Las llamadas de MPI se pueden clasificar en:

- Llamadas utilizadas para inicializar, administrar y finalizar comunicaciones: permiten inicializar la biblioteca de paso de mensajes, identificar el número de procesos (*size*) y el rango de los procesos (*rank*), donde las funciones principales son:
  - MPI_Init: primera llamada para cada proceso que establece un entorno.
  - MPI_Comm_size: devuelve el número de procesos del comunicador.
  - MPI_Comm_rank: devuelve el identificador del proceso dentro del comunicador.
  - MPI_Finalize: Termina la ejecución en MPI y libera los recursos utilizados.
- Llamadas utilizadas para transferir datos entre un par de procesos: operaciones de comunicación punto a punto, para envío y recepción, consiguiéndose mediante las llamadas:
  - MPI_Send: envía información de un proceso a otro.

- MPI_Recv: recibe información desde otro proceso.

Existe también la versión asíncrona o no bloqueante de estas llamadas las cuales son MPI_Isend y MPI_Irecv.

- Llamadas para transferir datos entre varios procesos: llamadas de comunicación grupales, donde alguna de estas llamadas son:
  - MPI_Barrier: sincronización global entre los procesadores del comunicador.
  - MPI_Bcas: envió de datos desde un proceso (*root*) a todos los demás.
  - MPI_Gather: recolección de datos de todos los procesos en uno de ellos.
  - MPI_Scatter: distribución de datos de un proceso entre todos los demás.
  - MPI_Reduce: reducción de los datos de cada procesador, dejando el resultado en uno de ellos (*root*).

2.3.2. TIPOS DE DATOS EN MPI

Los tipos de datos MPI son patrones de acceso de datos en memoria o en fichero. Pueden expresar patrones regulares o irregulares, con o sin huecos entre los datos.

Los tipos de datos básicos son los usados en los lenguajes de programación tradicionales como C: char (MPI_CHAR), byte (MPI_BYTE), integer (MPI_INT), float (MPI_FLOAT), etc. Los tipos de datos complejos son construidos a partir de tipos de datos básicos o recurrentemente de otros tipos de datos complejos. Como tipos de datos complejos podemos tener vectores y tipos estructurados.

Los tipos de datos complejos pueden ser representados como un árbol: los nodos internos son otros tipos de datos complejos usados en el proceso de construcción y las hojas son tipos de datos básicos. Los tipos de datos MPI son objetos opacos, p.ej. su estructura interna es directamente accesible al usuario. Una vez construido, el usuario

puede tener acceso a los tipos de datos por medio de un manejador. MPI ofrece un mecanismo para descifrar la composición de los tipos de datos (MPI_Type_get_envelope).

2.3.3. MPI-IO

MPI-IO se desarrolló en 1994 en el laboratorio Watson de IBM con el fin de proporcionar paralelismo a la E/S en aplicaciones desarrolladas en MPI. La Figura 2 muestra la arquitectura de ROMIO [18], la implementación más extendida de MPI-IO.

Las características más importantes de MPI-IO son:

- Portabilidad
  - Proporciona una interfaz estándar para las E/S, algo que no existía.
  - Interfaces UNIX tradicionales no estaban bien adaptados al paralelismo de E/S.
  - Plataforma/proveedor específico paralelo para E/S.
- Facilidad de uso
  - Escribir es como enviar y leer es como recibir.
  - Versatilidad de datos de MPI.
- Eficiencia/rendimiento potencial
  - Operaciones colectivas para permitir una mejora del potencial y del rendimiento.
  - Proporciona "hints" para posibles optimizaciones de E/S.
- Archivo de interoperabilidad
  - Proporciona un archivo estándar para la representación de datos.
  - Definición de un tratado universal de representación de datos externos.
- Apoyo a las representaciones definidas por los usuarios (tipos de datos, vistas, etc.).

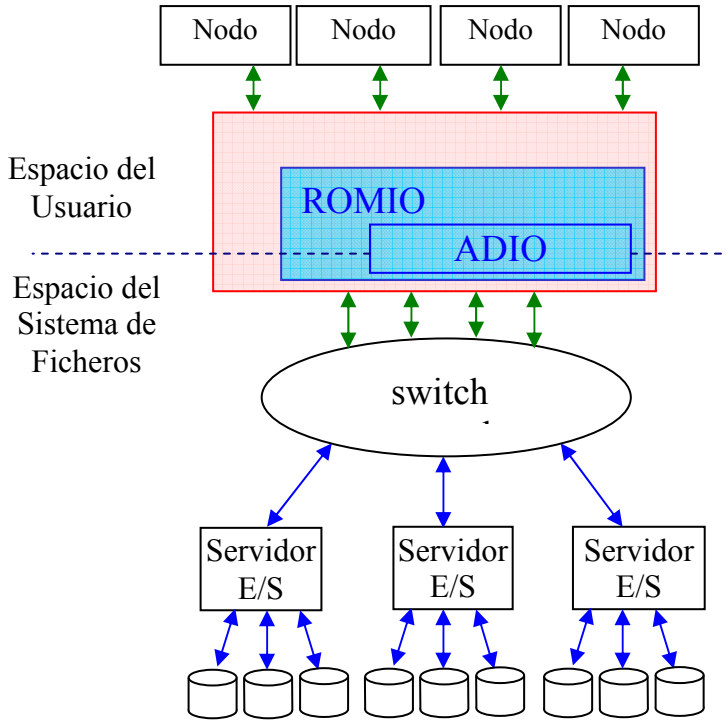


Figura 2: Arquitectura MPI-IO

2.3.4. MODELO DE FICHERO EN MPI-IO

Un fichero de MPI-IO es una colección ordenada de datos escritos en una máquina. Un fichero es abierto por un conjunto de procesos representados por un comunicador, donde todas las invocaciones de E/S colectiva o no, acceden al fichero mediante este comunicador. El desplazamiento en un fichero, es una posición absoluta medida en bytes en relación al principio del fichero, definiendo este el principio de la vista.

Un *etype* (tipo de datos elemental) es la unidad de acceso de datos (lectura y escritura) y desplazamiento (expresado como cuenta de *etypes*) pudiendo ser cualquier tipo de MPI predefinido. Un *filetype* es la base para dividir un archivo entre procesos y define una plantilla para tener acceso al fichero. Un *filetype* está compuesto por un *etype* o por un tipo de dato compuesto por varios *etypes*.

Una vista se define como el acceso actual de los datos visibles y accesibles de un fichero abierto como un conjunto ordenado de *etypes*. Cada proceso tiene su propia vista sobre el fichero, definida por tres valores: un desplazamiento, un *etype*, y un *filetype*. El patrón descrito por un *filetype* es repetitivo y su comienzo está definido por el valor del desplazamiento. La vista por defecto es una cadena de bytes (el desplazamiento es el cero, *etype* y *filetype* son igual a MPI_BYTE), p. ej. la vista traza una correspondencia de uno a uno al fichero. Un grupo de procesos puede usar vistas complementarias para alcanzar una distribución de datos global como un modelo dispersión/recolección. Un desplazamiento es una posición dentro de la vista, expresado como un número de *etypes*. Los huecos en el *filetype* de la vista son calculados mediante esta posición. El desplazamiento 0 es la posición del primer *etype* visible de la vista (después de saltar el desplazamiento y cualquier hueco inicial en la vista). Todas las operaciones sobre un fichero abierto usan el descriptor de fichero como una referencia al fichero.

2.3.4.1. ACCESO A LOS DATOS EN MPI-IO

En MPI-IO los datos se transmiten entre ficheros y procesos mediante llamadas de escritura y lectura. Las rutinas de acceso de datos pueden ser individuales o colectivas.

En la implementación de una rutina colectiva, a diferencia de las rutinas individuales, los procesos se coordinan entre sí para optimizar el acceso al dispositivo de E/S, esto se realiza mediante la combinación de pequeñas peticiones individuales en los nodos de cómputo en peticiones más grandes, optimizando así el funcionamiento de disco y red (reduciendo el número de transferencias necesarias).

Dependiendo del lugar donde se realiza la combinación, uno puede identificar dos métodos de E/S colectivas. Si la combinación ocurre en nodos intermedios o en los nodos de cómputo los llamaremos métodos E/S de dos fases.

2.3.5. ARQUITECTURA DE ROMIO

La implementación más extendida del estándar de MPI-IO es ROMIO el cual forma parte de distribuciones como MPICH [20], LAM, CV-MPI, NEC-MPI, y SGI-MPI. ROMIO, el interfaz de MPI-IO, es implementado sobre una interfaz abstracta llamada ADIO, siendo este el sistema de ficheros independiente. Sobre la cima de ADIO y bajo el interfaz de MPI-IO, ROMIO pone en práctica mecanismos como vistas, y varias optimizaciones de acceso como la E/S colectiva de dos fases (Two-phase I/O) [22].

El modelo de fichero de MPI-IO y las operaciones de ficheros están basados en los tipos de datos de MPI, donde una de las operaciones de fichero más interesantes es la declaración de una vista. Una vista ofrece varias ventajas: los datos no contiguamente almacenados son "vistos" contiguamente, facilitando la tarea del programador y permitiendo optimizaciones de E/S no contiguas. Al mismo tiempo una vista es una "pista" sobre el futuro modelo de acceso, pudiendo ser usado para optimizar el acceso.

La interfaz de ADIO contiene las típicas funciones para manejar ficheros: open, close, fcntl, read, write, etc. En la implementación actual de ADIO el mecanismo de vistas reside en la capa de MPI-IO, como se muestra en la Figura 3. La vista mapea sobre un espacio lineal del fichero mediante MPI-IO y, a su vez, el fichero sobre subficheros o discos por el sistema de ficheros. Los dos mapeos se realizan de forma explícita, incluso cuando la vista mapea datos contiguamente posicionados sobre un subfichero/disco. ROMIO contiene una implementación de la técnica two-phase I/O, anteriormente descrita. La fase de mezcla es implementada en la capa de MPI-IO. La fase de E/S transfiere el buffer colectivo del nodo de cómputo al sistema de ficheros. La fase de E/S es implementada en una función de ADIO, que a su vez llama funciones de acceso de sistema de ficheros.

La técnica two-phase I/O de ROMIO puede necesitar hasta tres transferencias de red para superponer una región existente. La primera corresponde a la fase de intercambio, la segunda la transacción "leído modifica escriben" necesaria para

actualizar el fichero. La parte leída puede ser optimizada cuando la región del fichero de huecos es superpuesta.

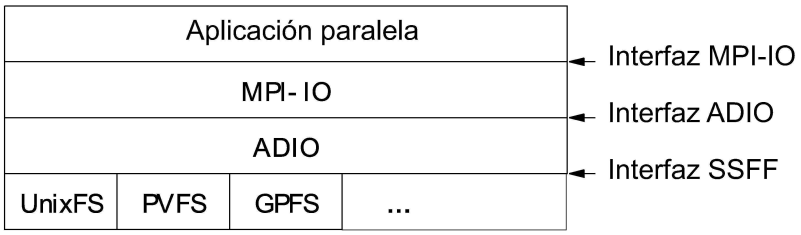


Figura 3: Arquitectura ROMIO

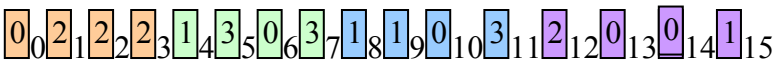
2.3.5.1. TÉCNICA TWO-PHASE I/O

El uso de la técnica two-phase I/O es una de las técnicas usadas a la hora de distribuir las fracciones de datos que deberán ser escritas en cada uno de los procesos que quieran hacer uso del fichero distribuido. Algunos de los sistemas distribuidos que pueden hacer uso de esta técnica son AFS, xFS, GPFS, o PVFS, por poner algunos ejemplos. Además, dichos sistemas pueden usar interfaces POSIX o MPI-IO. En el presente apartado se hablará de la técnica two-phase I/O.

La técnica two-phase I/O para la escritura/lectura colectiva de datos no contiguos, debe su nombre a que se realiza en dos fases: la primera fase consiste en el intercambio de datos entre los procesos que intervienen en la operación. La segunda fase es la escritura o lectura de los datos en disco.

En primer lugar, lo que hace la técnica two-phase I/O, es asignar qué parte del fichero distribuido corresponderá a cada proceso que quiera tener acceso a dicho fichero. La técnica two-phase I/O asignará cada parte del fichero distribuido a cada proceso de acuerdo a su posición, es decir, la primera parte será para el primer proceso, la segunda parte, para el segundo proceso… y así sucesivamente.

Ejemplo. Se dará la siguiente distribución para cuatro procesos accediendo a un fichero distribuido, usando la técnica two-phase I/O:



n_m : Dato situado en la posición m del fichero distribuido, asignado al proceso n.

Cada proceso dispondrá de los siguientes datos:

Proceso 0: 0 0 0 6 0 10 0 13 0 14

Proceso 1: 1 4 1 8 1 9 1 15

Proceso 2: 2 1 2 2 2 3 2 12

Proceso 3: 3 5 3 7 3 11

Según el color, cada parte del fichero se asigna a un proceso:

al 0, al 1, al 2, al 3.

Así pues, cuando se llegue a la fase de intercambio de datos, para construir el fichero distribuido, se realizarán las siguientes comunicaciones:

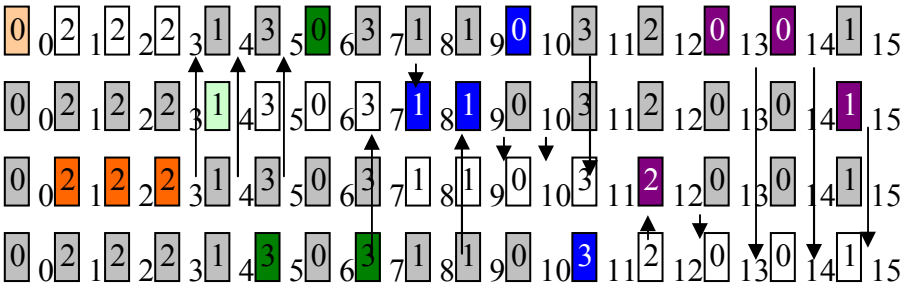


Figura 4: Ejemplo de intercambio de datos con two-phase I/O

En la primera de las fases, el intercambio de datos, todos los procesos mandarán los datos que se encuentren en partes del fichero distribuido asignadas a otros procesos, al proceso al que pertenecen. Es en este momento en el que realizan los intercambios de datos por la red, y en el que interviene la técnica para distribuir los datos correspondientes entre los distintos procesos. Así pues, tras esta fase, todos los procesos deberían tener un buffer con todos los datos que necesitan en su parte del fichero distribuido. En la segunda de las fases, la escritura de datos, todos los procesos escribirán su parte de datos del fichero distribuido en disco.

2.4. MPE

MPE intenta proveer a los programadores de instrumentos de análisis que permitan evaluar el funcionamiento de sus programas MPI. MPE se basada en el aprovechamiento del post procesamiento y de la generación de trazas. Estas herramientas incluyen librerías con un conjunto de programas auxiliares, y un conjunto de instrumentos de visualización gráfica.

El primer conjunto de herramientas están diseñadas para ser usadas en programas MPI, es decir, las bibliotecas que proporcionan una colección de rutinas para crea ficheros de traza (log). Estos ficheros log pueden ser creados a mano insertando llamadas de MPE en el programa MPI, o automáticamente mediante la unión de las librerías apropiadas de MPE. Otra solución disponible es combinar los dos métodos anteriores. Corrientemente, el MPE ofrece a las 3 soluciones las siguientes librerías:

- **Librería de trazas:** Muestra todas las llamadas de MPI. Cada llamada de MPI está precedida por una línea que contiene la fila en MPI_COMM_WORLD de la llamada del proceso, y seguido de otra línea que indica que la llamada se ha completado. El envío y recepción de rutinas también indica los valores de la cantidad de datos que envía o recibe, etiqueta, y el compañero (el destino para el que envía, la fuente para el que recibe). La salida es a la salida estándar.

- **Librería de animación:** Esto es una forma simple de animación de programa en tiempo real y requiere rutinas X-Window.
- **Librería de logging:** Esta es la librería más útil de las que tiene MPE. Forman la base para generar ficheros de log en los programas MPI que esté utilizando el usuario. Actualmente hay 3 formatos de ficheros de log diferentes permitidos en MPE. El formato del fichero log por defecto es CLOG. Este es básicamente una colección de eventos con fecha. También está ALOG que proporciona compatibilidad, pero que está siendo desarrollado. Y el más poderoso es SLOG, el formato de LOGFILE Escalable, que puede ser convertido a partir de CLOG, una vez que se ha generado éste, o puede ser generado directamente cuando el programa MPI es ejecutado. El juego de programas auxiliares en MPE incluye el convertidor de formato de log (p.ej. clog2slog), la impresión (p.ej. slog_print)… Normalmente, los instrumentos gráficos de MPE incluyen 3 programas de demostración, resultados para ALOG, jumpshot-2 para CLOG y jumpshot-3 para la SLOG.

MPE puede ser configurado e instalado como una extensión a la mayor parte de las implementaciones de MPI, p.ej. MPICH, LAM, MPI DEL SGI, CV-UX'S MPI y la versión de MPI de IBM. Ha sido integrado a la perfección en la distribución MPICH, por lo que MPE será instalado automáticamente durante el proceso de instalación de MPICH.

2.5. JUMPSHOT

Jumpshot es la herramienta de análisis de la ejecución de programas paralelos que se distribuye junto con MPICH. Es una herramienta compleja, que permite analizar gráficamente ficheros de trazas (de tipo log) obtenidos de la ejecución de programas MPI en los que previamente se han añadido una serie de llamadas a MPE para recoger información.

Para generar el fichero log podemos añadir puntos de muestreo en lugares concretos (añadiendo funciones de MPE), o, en casos sencillos, tomar datos de todas las funciones MPI.

Una vez compilado, lo ejecutamos y obtenemos un fichero de trazas, de tipo clog2. Ahora podemos ejecutar jumpshot para analizar el fichero de trazas obtenido. Jumphot4 trabaja con un formato de tipo slog2, por lo que previamente hará una conversión a dicho formato. Si se desea, se puede hacer esa conversión ejecutando clog2TOslog2.

La ventana inicial de jumpshot nos permite escoger el fichero que queremos visualizar.

En una nueva ventana aparece un esquema gráfico de la ejecución, en el que las diferentes funciones de MPI se representan con diferentes colores. En el caso de las comunicaciones punto a punto, una flecha une la emisión y recepción de cada mensaje. Con el botón derecho del ratón podemos obtener datos de las funciones ejecutadas y de los mensajes transmitidos.

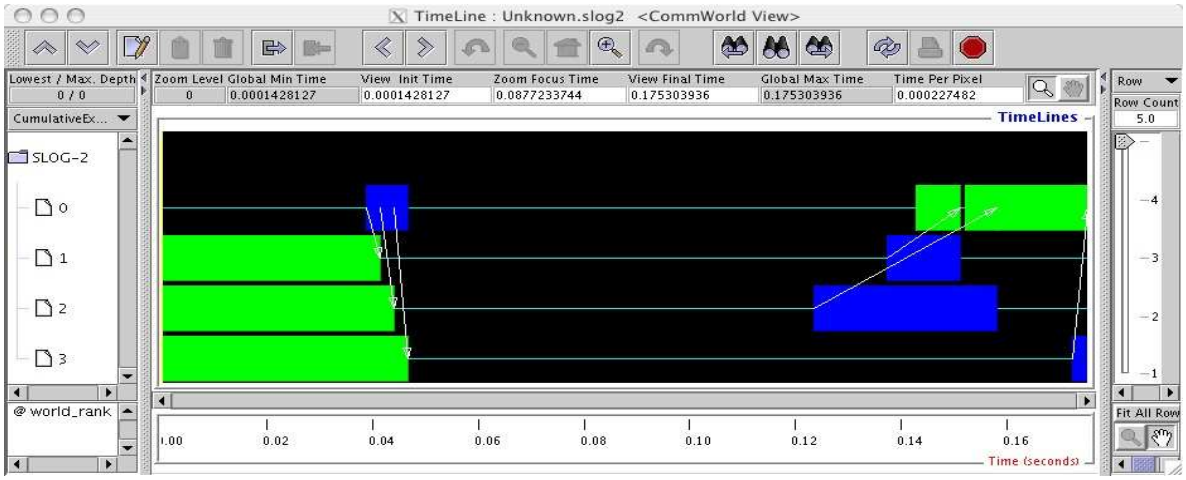


Figura 5: Ejemplo de ejecución de Jumpshot

2.6. WINDOWS HPC SERVER 2008

Publicado por Microsoft en septiembre de 2008, es el sucesor del producto de Windows Compute Cluster Server 2003. Al igual que WCCS, Windows HPC Server 2008 está diseñado para aplicaciones de gama alta que requieren agrupaciones informáticas de alto rendimiento. Esta versión del servidor escala de manera eficiente a miles de núcleos, e incluye características exclusivas de trabajo HPC: una nueva de alta velocidad NetworkDirect RDMA, alta eficiencia, escalabilidad y herramientas de gestión de grupo, una arquitectura orientada a servicios (SOA) planificador de tareas, y la interoperabilidad entre cluster a través de estándares como *High Performance Computing Basic Profile* (HPCBP) especificado por Open Grid Forum (OGF).

En junio de 2008, un sistema construido en colaboración con el Centro Nacional para Aplicaciones de Supercomputación (NCSA) y Microsoft clasificó en el puesto 23 en la lista Top500, con una puntuación de Linpack de 68,5 teraflops.

En noviembre de 2008 en los rankings publicados por Top500, un equipo con el sistema Windows HPC construido por el Shanghai Supercomputer Center alcanza los 180,6 teraflops, que sitúa al sistema en el puesto 10 de la clasificación de los superordenadores más rápidos del mundo.

2.7. NTFS

NTFS (*New Technology File System*), es un sistema de archivos diseñado específicamente para Windows NT, y utilizado por las versiones recientes del sistema operativo Windows. Ha reemplazado al sistema FAT utilizado en versiones antiguas de Windows y en DOS.

Fue creado para lograr un sistema de archivos eficiente y seguro y está basado en el sistema de archivos HPFS de IBM/Microsoft usado en el sistema operativo OS/2. También tiene características del sistema de ficheros HFS diseñado por Apple. NTFS permite definir el tamaño del clúster de forma independiente al tamaño de la partición. El tamaño mínimo del bloque es de 512 bytes. Este sistema también admite compresión nativa de archivos y encriptación. Es un sistema ideal para particiones de gran tamaño, pudiendo manejar discos de hasta 2 terabytes. Windows NT, 2000, 2003, XP y Vista soportan el sistema NTFS.

Sus desventajas son:

- Utiliza gran cantidad de espacio en disco para sí mismo.
- No es compatible con sistemas operativos como DOS, Windows 95, 98 ni ME.
- No puede ser usado en disquetes.
- La conversión a NTFS es unidireccional, por lo tanto, no se puede volver a convertir en FAT al actualizar la unidad.

Sus ventajas y mejoras con respecto al sistema FAT son:

- Compatibilidad mejorada con los metadatos.
- Uso de estructura de datos avanzadas (árboles-B), optimizando el rendimiento, estabilidad y aprovechando espacio en disco, pues acelera el acceso a los ficheros y reduce la fragmentación.
- Mejora de la seguridad.
- Listas de control de acceso

- El registro de transacciones (journaling), que garantiza la integridad del sistema de ficheros.

Existen tres versiones de NTFS: v1.2 en NT 3.51 y NT 4, v3.0 en Windows 2000 y v3.1 en Windows XP y Windows 2003 Server. Los detalles de la implementación son secretos de Microsoft.

2.8. SAMBA

Samba es una implementación libre del protocolo de archivos compartidos de Microsoft Windows® (antiguamente llamado SMB, renombrado recientemente a CIFS) para sistemas de tipo UNIX. De esta forma, es posible que ordenadores con GNU/Linux, Mac OS X o Unix en general se vean como servidores o actúen como clientes en redes de Windows®. Samba también permite validar usuarios haciendo de Controlador Principal de Dominio (PDC), como miembro de dominio e incluso como un dominio *Active Directory* para redes basadas en Windows; aparte de ser capaz de servir colas de impresión, directorios compartidos y autentificar con su propio archivo de usuarios.

Samba es una implementación de una docena de servicios y una docena de protocolos, entre los que están: NetBIOS sobre TCP/IP (NetBT), SMB (también conocido como CIFS), DCE/RPC o más concretamente, MSRPC, el servidor WINS también conocido como el servidor de nombres NetBIOS (NBNS), la suite de protocolos del dominio NT, con su Logon de entrada a dominio, la base de datos del gestor de cuentas seguras (SAM), el servicio Local Security Authority (LSA) o autoridad de seguridad local, el servicio de impresoras de NT y recientemente el Logon de entrada de Active Directory, que incluye una versión modificada de Kerberos y una versión modificada de LDAP. Todos estos servicios y protocolos son frecuentemente referidos de un modo incorrecto como NetBIOS o SMB.

Samba configura directorios Unix/Linux (incluyendo sus subdirectorios) como recursos para compartir a través de la red. Para los usuarios de Microsoft Windows, estos recursos aparecen como carpetas normales de red. Los usuarios de Linux pueden montar en sus sistemas de archivos estas unidades de red como si fueran dispositivos locales, o utilizar la orden smbclient para conectarse a ellas muy al estilo del cliente de la línea de órdenes ftp. Cada directorio puede tener diferentes permisos de acceso sobrepuestos a las protecciones del sistema de archivos que se esté usando en Linux.

3. PROGRAMACIÓN EN .NET FRAMEWORK

.NET es un proyecto de Microsoft para crear una nueva plataforma de desarrollo de software con énfasis en transparencia de redes, con independencia de plataforma y que permita un rápido desarrollo de aplicaciones. .NET se podría considerar como una respuesta de Microsoft al creciente mercado de los negocios en entornos Web, en competencia con la plataforma Java de Sun.

A largo plazo Microsoft pretende reemplazar la Interfaz de Programación de Aplicaciones (API por sus siglas en inglés) Win32 o Windows API con la plataforma .NET. Esto debido a que la API Win32 o Windows API fue desarrollada sobre la marcha, careciendo de documentación detallada, uniformidad y cohesión entre sus distintos componentes, provocando múltiples problemas en el desarrollo de aplicaciones para el sistema operativo Windows. La plataforma .NET pretende solventar la mayoría

de estos problemas proveyendo un conjunto único y expandible con facilidad, de bloques interconectados, diseñados de forma uniforme y bien documentados, que permitan a los desarrolladores tener a mano todo lo que necesitan para producir aplicaciones sólidas.

Debido a las ventajas que la disponibilidad de una plataforma de este tipo puede darle a las empresas de tecnología y al público en general, muchas otras empresas e instituciones se han unido a Microsoft en el desarrollo y fortalecimiento de la plataforma .Net, ya sea por medio de la implementación de la plataforma para otros sistemas operativos aparte de Windows (Proyecto Mono de Ximian/Novell para Linux/MacOS X/BSD/Solaris), el desarrollo de lenguajes de programación adicionales para la plataforma (ANSI C de la Universidad de Princeton, NetCOBOL de Fujitsu, Delphi de Borland, entre otros) o la creación de bloques adicionales para la plataforma (como controles, componentes y librerías de clases adicionales); siendo algunas de ellas iniciativas de distribución gratuita bajo la licencia GNU.

Actualmente, el Framework de .Net es una plataforma no incluida en los diferentes sistemas operativos distribuidos por Microsoft, por lo que es necesaria su instalación previa a la ejecución de programas creados mediante .Net. El Framework se puede descargar gratuitamente desde la Web oficial de Microsoft.

3.1 LA ARQUITECTURA DE .NET FRAMEWORK

Es la arquitectura básica de la plataforma .Net. El Framework de .Net es una infraestructura sobre la que se reúne todo un conjunto de lenguajes y servicios que simplifican el desarrollo de aplicaciones. Mediante esta herramienta se ofrece un entorno de ejecución distribuido.

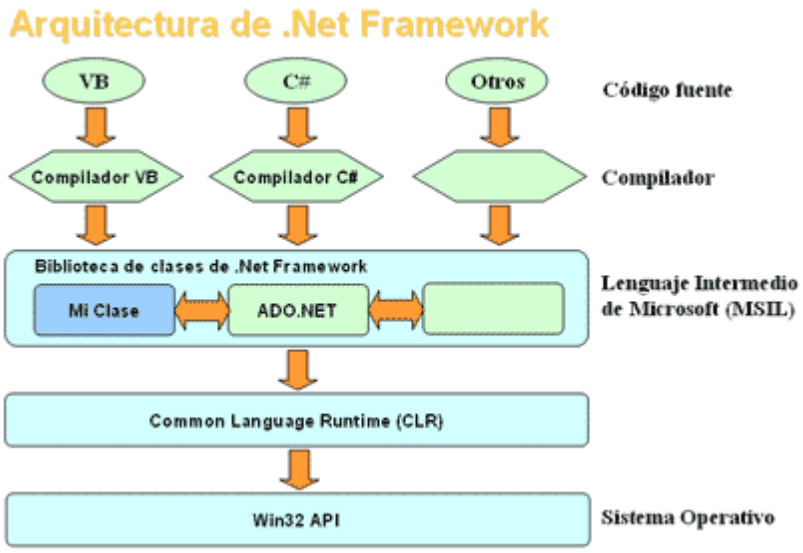


Figura 6: La arquitectura de .Net FrameWork

Los principales componentes de este entorno son:

- Lenguajes de compilación
- Biblioteca de clases de .Net
- CLR (*Common Language Runtime*)

.Net Framework soporta múltiples lenguajes de programación y, es posible desarrollar cualquier tipo de aplicación con cualquiera de estos lenguajes. Lenguajes incluidos en la plataforma: C# (C *Sharp*), Visual Basic, C++, J# (Java #), etc.

3.1.1 COMMON LANGUAGE RUNTIME

El CLR es el verdadero núcleo del Framework de .Net, entorno de ejecución en el que se cargan las aplicaciones desarrolladas en los distintos lenguajes, ampliando el conjunto de servicios del sistema operativo (W2k y W2003).

La herramienta de desarrollo compila el código fuente de cualquiera de los lenguajes soportados por .Net en un código intermedio (MSIL, Microsoft Intermediate Lenguaje), similar al BYTECODE de Java. Para generar dicho código el compilador se

basa en el *Common Language Specification* (CLS) que determina las reglas necesarias para crear ese código MSIL compatible con el CLR.

Para ejecutarse se necesita un segundo paso, un compilador JIT (*Just-In-Time*) es el que genera el código máquina real que se ejecuta en la plataforma del cliente. De esta forma se consigue con .Net independencia de la plataforma hardware, que no de sistema operativo.

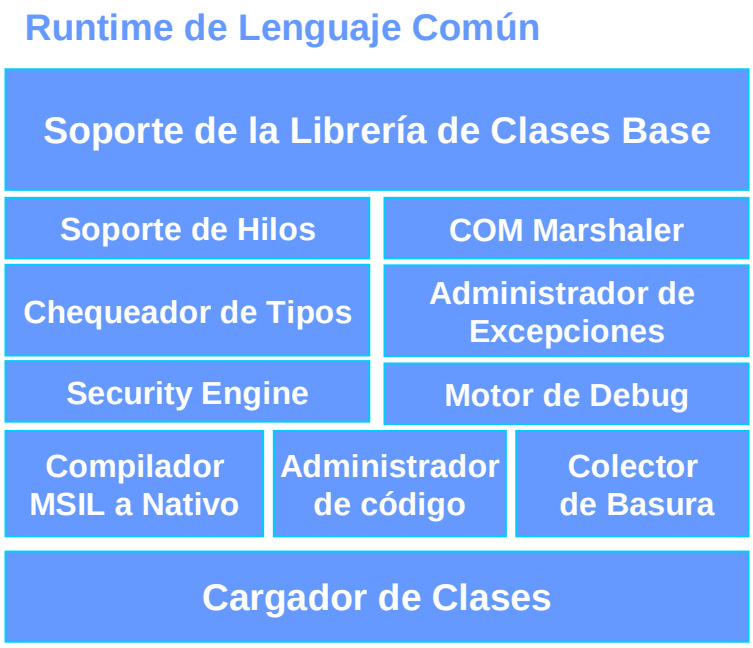


Figura 7: Runtime de Lenguaje Común

La compilación JIT la realiza el CLR a medida que el programa invoca métodos, el código ejecutable obtenido, se almacena en la memoria caché del ordenador, siendo recompilado de nuevo sólo en el caso de producirse algún cambio en el código fuente.

3.1.2 BIBLIOTECAS DE CLASES

Las clases base proporcionan funciones estándar, como las de entrada/salida, manipulación de cadenas, administración de seguridad, comunicaciones en red, administración de subprocesos, administración de textos y funciones de diseño de la interfaz de usuario.

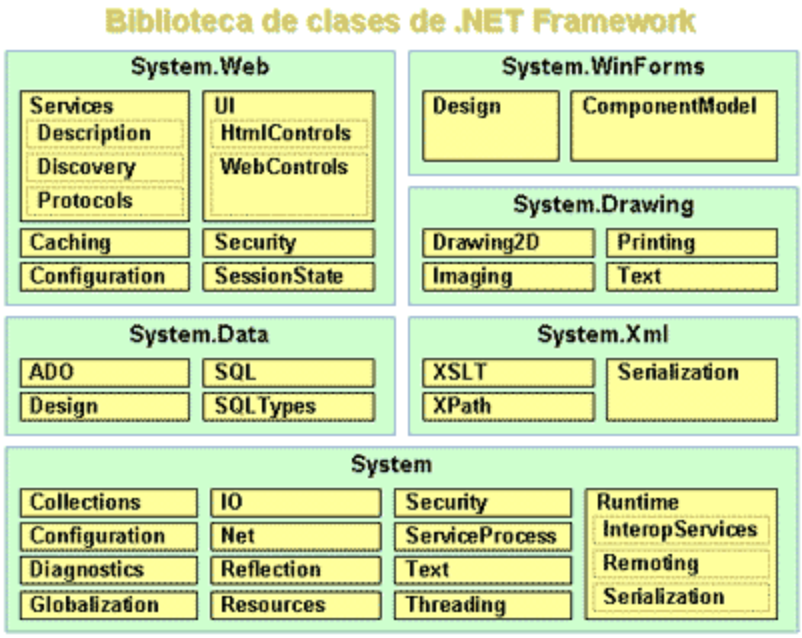


Figura 8: Biblioteca de clases de .Net Framework

Las clases de ADO.NET permiten a los programadores interactuar con los datos obtenidos en formato XML a través de las interfaces OLE DB, ODBC, Oracle y SQL Server. Las clases XML permiten la manipulación, búsqueda y conversión de objetos XML. Las clases ASP.NET son compatibles con el desarrollo de aplicaciones basadas en Web y de servicios Web XML. Las clases de Windows Forms son compatibles con la generación de aplicaciones cliente inteligentes basadas en escritorio. En conjunto, las bibliotecas de clases ofrecen una interfaz de desarrollo común y coherente en todos los lenguajes compatibles con Windows .NET Framework.

La forma de organizar la biblioteca de clases de .Net dentro del código es a través de los espacios de nombres (namespaces), donde cada clase está organizada en espacios de nombres según su funcionalidad. Por ejemplo, para manejar ficheros se utiliza el espacio de nombres System.IO y si lo que se quiere es obtener información de una fuente de datos se utilizará el espacio de nombres System.Data.

La principal ventaja de los espacios de nombres de .Net es que de esta forma se tiene toda la biblioteca de clases de .Net centralizada bajo el mismo espacio de nombres (System). Además, desde cualquier lenguaje se usa la misma sintaxis de invocación, ya que a todos los lenguajes se aplica la misma biblioteca de clases.

3.1.3 ENSAMBLADOS

Uno de los mayores problemas de las aplicaciones actuales es que en muchos casos tienen que tratar con diferentes archivos binarios (DLL´s), elementos de registro, conectividad abierta a bases de datos (ODBC), etc.

Para solucionarlo el Framework de .Net maneja un nuevo concepto denominado ensamblado. Los ensamblados son ficheros con forma de EXE o DLL que contienen toda la funcionalidad de la aplicación de forma encapsulada. Por tanto la solución al problema puede ser tan fácil como copiar todos los ensamblados en el directorio de la aplicación.

Con los ensamblados ya no es necesario registrar los componentes de la aplicación. Esto se debe a que los ensamblados almacenan dentro de sí mismos toda la información necesaria en lo que se denomina el manifiesto del ensamblado. El manifiesto recoge todos los métodos y propiedades en forma de meta-datos junto con otra información descriptiva, como permisos, dependencias, etc.

Para gestionar el uso que hacen la aplicaciones de los ensamblados, .Net utiliza la llamada caché global de ensamblados (GAC, Global Assembly Cache). Así, .Net Framework puede albergar en el GAC los ensamblados que puedan ser usados por varias aplicaciones e incluso distintas versiones de un mismo ensamblado, algo que no era posible con el anterior modelo COM.

3.2 VENTAJAS E INCONVENIENTES DE LA PLATAFORMA .NET

A continuación se resumen las ventajas más importantes que proporciona .Net Framework:

- Código administrado: El CLR realiza un control automático del código para que este sea seguro, es decir, controla los recursos del sistema para que la aplicación se ejecute correctamente.
- Interoperabilidad multilenguaje: El código puede ser escrito en cualquier lenguaje compatible con .Net ya que siempre se compila en código intermedio (MSIL).
- Compilación *just-in-time*: El compilador JIT incluido en el Framework compila el código intermedio (MSIL) generando el código máquina propio de la plataforma. Se aumenta así el rendimiento de la aplicación al ser específico para cada plataforma.
- *Garbage collector*: El CLR proporciona un sistema automático de administración de memoria denominado recolector de basura (*garbage collector*). El CLR detecta cuándo el programa deja de utilizar la memoria y la libera automáticamente. De esta forma el programador no tiene por que liberar la memoria de forma explícita aunque también sea posible hacerlo manualmente (mediante el método *dispose()* liberamos el objeto para que el recolector de basura lo elimine de memoria).
- Seguridad de acceso al código: Se puede especificar que una pieza de código tenga permisos de lectura de archivos pero no de escritura. Es posible aplicar distintos niveles de seguridad al código, de forma que se puede ejecutar código procedente del Web sin tener que preocuparse si esto va a estropear el sistema.
- Despliegue: Por medio de los ensamblados resulta mucho más fácil el desarrollo de aplicaciones distribuidas y el mantenimiento de las mismas. El Framework realiza esta tarea de forma automática mejorando el rendimiento y asegurando el funcionamiento correcto de todas las aplicaciones.

A continuación se resumen las desventajas más importantes que proporciona .Net Framework:

- Procesos como la recolección de basura de .Net o la administración de código introducen factores de sobrecarga que repercuten en la demanda de más requisitos del sistema.
- El código administrado proporciona una mayor velocidad de desarrollo y mayor seguridad de que el código sea bueno. En contrapartida el consumo de recursos durante la ejecución es mucho mayor, aunque con los procesadores actuales esto cada vez es menos inconveniente.
- El nivel de administración del código dependerá en gran medida del lenguaje que utilicemos para programar. Por ejemplo, mientras que Visual Basic .Net es un lenguaje totalmente administrado, C# permite la administración de código de forma manual, siendo por defecto también un lenguaje administrado. Mientras que C++ es un lenguaje no administrado en el que se tiene un control mucho mayor del uso de la memoria que hace la aplicación.

4. HERRAMIENTAS DE EVALUACIÓN DE E/S

Las herramientas de evaluación o *benchmarks* son unos sistemas utilizados para medir el rendimiento de un sistema o componente de un sistema, frecuentemente en comparación con el cual se refiere específicamente a la acción de ejecutar un *benchmark*. La palabra *benchmark* es un anglicismo traducible al castellano como *comparativa*. Si bien también puede encontrarse esta palabra haciendo referencia al significado original en la lengua anglosajona, es en el campo informático donde su uso está más ampliamente extendido. Más formalmente puede entenderse que un *benchmark* es el resultado de la ejecución de un programa informático o un conjunto de programas en una máquina, con el objetivo de estimar el rendimiento de un elemento concreto o la totalidad de la misma, y poder comparar los resultados con máquinas

similares. En términos de ordenadores, un *benchmark* podría ser realizado en cualquiera de sus componentes, ya sea CPU, RAM, tarjeta gráfica, etc.

La tarea de ejecutar un *benchmark* originalmente se reducía a estimar el tiempo de proceso que lleva la ejecución de un programa (medida por lo general en miles o millones de operaciones por segundo). Con el correr del tiempo, la mejora en los compiladores y la gran variedad de arquitecturas y situaciones existentes convirtieron a esta técnica en toda una especialidad. La elección de las condiciones bajo la cual dos sistemas distintos pueden compararse entre sí es especialmente ardua, y la publicación de los resultados suele ser objeto de candentes debates cuando éstos se abren a la comunidad.

También puede realizarse un "*benchmark* de software", es decir comparar el rendimiento de un software contra otro o de parte del mismo, por ejemplo, comparar distintas consultas a una base de datos para saber cuál es la más rápida o directamente partes de código.

El *Benchmark* es también un proceso continuo de medir productos, servicios y prácticas contra competidores más duros o aquellas compañías reconocidas como líderes en la industria.

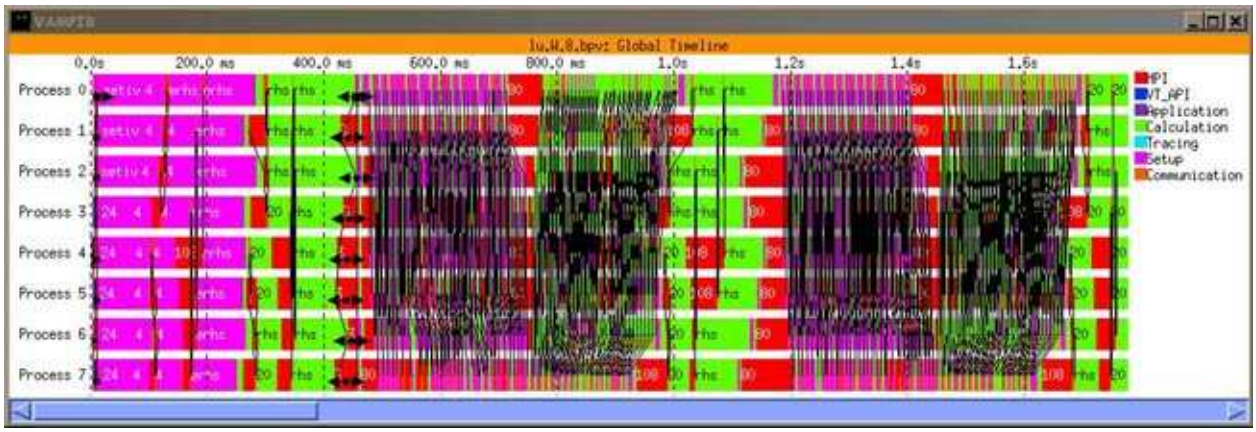


Figura 9: Ejemplo de benchmarks de rendimiento de un sistema

4.1 CARACTERÍSTICAS

Los benchmark tienen las siguientes funcionalidades:

- Comprobar si las especificaciones de los componentes están dentro del margen propio del mismo.
- Maximizar el rendimiento con un presupuesto dado.
- Minimizar costes manteniendo un nivel mínimo de rendimiento.
- Obtener la mejor relación coste/beneficio (con un presupuesto o unas exigencias dadas).
- Coayuda a lograr una posición más competitiva.

4.2 TIPOS DE BENCHMARKS

Dado el rendimiento objetivo, existen diferentes tipos de benchmark:

- **Sintéticos vs Aplicaciones**

    - Sintéticos: están especialmente diseñadas para medir el rendimiento de un componente individual de un ordenador, normalmente llevando el componente escogido a su máxima capacidad.
    - Aplicaciones: herramientas basadas en aplicaciones reales, simulan una carga de trabajo para medir el comportamiento global del equipo.

- **Bajo nivel vs Alto nivel**

  - Test de Bajo nivel: Miden directamente el rendimiento de los componentes Ejemplo: el reloj de la CPU, los tiempos de la DRAM y de la caché SRAM, tiempo de acceso medio al disco duro, latencia, tiempo de cambio de pista, etc.

  - Test de Alto nivel: Están más enfocados a medir el rendimiento de la combinación componente/controlador/SO de un aspecto específico del sistema, como por ejemplo el rendimiento de E/S con ficheros, o el rendimiento de una determinada combinación de componentes/controlador/SO/aplicación. Ejemplo: Velocidad de compresión zip

4.3 COLLPERF

Collperf es un benchmark de prueba integrado dentro de la distribución de ROMIO, utilizado para estimar el funcionamiento de las rutinas de E/S colectivas de MPI-IO. La prueba mide la tasa de transferencia de E/S tanto para la escritura como para la lectura de un archivo, con un patrón acceso definido como una serie tridimensional distribuida por bloques.

Para la E/S colectiva de ROMIO, todos los nodos computan, y el tamaño del buffer colectivo es 16 MBytes (por defecto en la versión 1.0.8 de MPICH2). La división de datos se realiza por la asignación de un número de procesos sobre cada dimensión cartesiana.

4.4 MPI-TILE-IO

MPI-Tile-IO en un benchmark de MPI-IO, utilizado para testear el funcionamiento del acceso a datos no contiguos. Este sistema de evaluación, el acceso de E/S a los datos no contiguos es realizado en un único paso, mediante el uso de operaciones colectivas de E/S. Este benchmark es muy conocido dentro de la comunidad científica y referenciado en multitud de publicaciones.

El benchmark testea el rendimiento del acceso simultáneo a un array bidimensional de datos, mediante la simulación del tipo de trabajo existente en algunas aplicaciones visuales y numéricas.

El benchmark dispone de una gran cantidad de opciones de configuración, como el tamaño de datos, la distribución de los datos, solapamientos, etc.

4.5 SimParIO

SimParIO es una herramienta de evaluación que simula el comportamiento de una aplicación paralela. La principal meta de la evaluación SimParIO es facilitar la evaluación del grado de solapamiento entre computación y E/S paralela.

SimParIO consiste en la alternancia de las fases de computación y E/S. La fase de cómputo se simula mediante una parada activa (sleep). El tamaño de la fase de cómputo es configurable por el usuario. Cada fase de cómputo es seguida de una fase de E/S, en la cual, todos los procesos escriben o leen regiones del fichero que no se encuentran solapadas. El patrón de acceso puede ser contiguo o no contiguo.

El benchmark ofrece una gran cantidad de elementos configurables: el tamaño de acceso, el tipo de acceso al sistema de ficheros, tamaño de los buffer auxiliares, estimación del tiempo de cómputo entre otras.

5. SISTEMA DE E/S PARALELA AHPIOS

AHPIOS (*Ad-hoc parallel I/O System*), es el primer sistema de E/S paralela escalable completamente implementado en MPI (*Message Passing Interface*). En la actualidad la mayoría de los supercomputadores pertenecientes al Top500 utilizan sistemas de ficheros como PVFS, GPFS o Lustre usados mayoritariamente en aplicaciones de procesamiento de datos y visualización, pudiéndose beneficiar estas aplicaciones del sistema AHPIOS al ofrecerlas una solución transparente de E/S paralela, que casi no cuesta instalarla y mantiene un sistema de ficheros paralelo.

AHPIOS puede ser usado como un middleware alojado entre MPI-IO y los recursos de almacenamiento distribuido, proporcionando un alto rendimiento de acceso a los ficheros. AHPIOS puede ser usado como una alternativa ligera y barata a cualquier sistema de fichero paralelo.

Las principales características de AHPIOS son las siguientes:

- **Alto rendimiento**: se logra a través de la estrecha integración de MPI-IO y el sistema de almacenamiento, que permite un acceso eficiente a los datos, a través de dos niveles de caché cooperativa y una estrategia adquisición de datos asíncrona.

- **Escalabilidad**: el sistema es escalable con ambas jerarquías de memorias y con el almacenamiento. El primer nivel de caché es ajustable al número de procesos que participan en la aplicación paralela. El segundo nivel de caché y el sistema de almacenamiento global (disco) se adapta al número de servidores AHPIOS y recursos de almacenamiento.

- **Portabilidad**: logrado gracias a la completa implementación en MPI (siendo esta la primera implementación conocida de un sistema paralelo de E/S en MPI).

- **Simplicidad**: es fácil de usar y no es necesaria la modificación de las aplicaciones existentes. El funcionamiento de AHPIOS puede ser configurada por el usuario a partir de un fichero de texto plano.

5.1 VISIÓN GENERAL

Dada una aplicación en MPI que acede a distintos ficheros a través de la interfaz de MPI-IO, y un conjunto de recursos de almacenamiento distribuidos, AHPIOS construye bajo demanda una partición distribuida, la cual puede ser accedida de forma transparente y eficiente. En cada partición AHPIOS, los usuarios pueden crear un espacio de nombres de directorio, de la misma manera que en cualquier sistema de ficheros. Los ficheros almacenados en una partición AHPIOS son listados de forma transparente sobre los recursos de almacenamiento al igual que en cualquier sistema de ficheros paralelo. Estas particiones son gestionadas por un conjunto de servidores de almacenamiento, los cuales, se ejecutan como una aplicación MPI independiente. El

acceso a una partición AHPIOS es realizado a través de la interfaz de MPI-IO. Una partición puede ser construida escalando hacia arriba o hacia abajo, bajo demanda mientras la aplicación se esté ejecutando.

El sistema maneja una jerarquía de memorias caché cooperativas, como se muestra en la Figura 10:

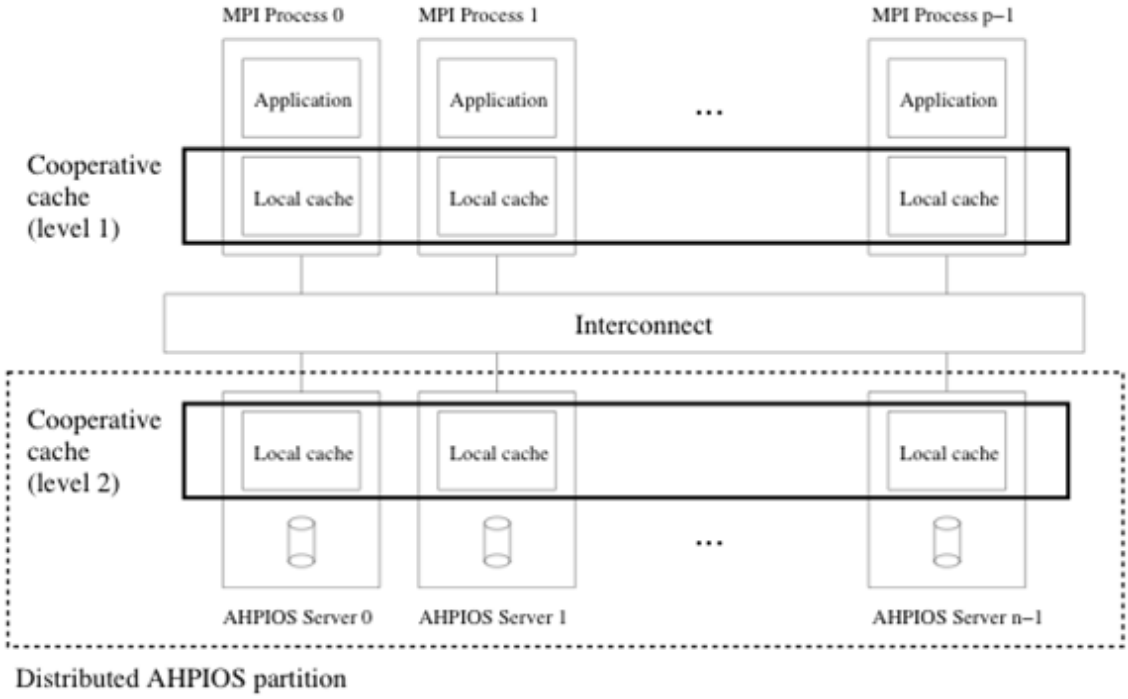


Figura 10: Esquema AHPIOS

En primer lugar las cachés de las aplicaciones clientes se agrupan en un buffer colectivo, el cual será usado por la capa de MPI-IO para reordenar y recopilar las peticiones de E/S con el fin de mejorar el rendimiento de acceso para la E/S. En segundo lugar los servidores, AHPIOS gestiona una copia de caché cooperativa. La comunicación interna y entre estas capas se realiza a través de operaciones estándar de MPI.

Una estrategia de adquisición de datos, oculta la latencia de transferencia de bloques de datos entre los niveles de la jerarquía de caches. La transferencia de datos entre la cache del primer nivel (clientes) y la cache del segundo nivel (servidores), así como entre la caché de los servidores y el almacenamiento final, se realizan de forma asíncrona.

Los mecanismos de MPI-IO y las optimizaciones como las *vistas* y el grupo de E/S están fuertemente integrados dentro del sistema de almacenamiento. Las *vistas* pueden ser aplicadas, ya sea en cliente o en el servidor y el grupo de buffers debe residir en el cliente o cerca del almacenamiento como en los nodos de los servidores.

Para cada ejecución, el usuario puede definir la configuración de los parámetros de manera centralizada, como pueden ser el número de recursos de almacenamiento, el tamaño del buffer de red y del tamaño bloque, el tamaño de la caché del cliente y del servidor, el tipo de política de planificación, etc. El usuario tiene el control de la configuración de MPI-IO y del sistema paralelo E/S desde un único fichero.

El sistema puede consistir en varias particiones independientes, donde cada una es manejada por diferentes grupos de servidores, con distintas configuraciones.

5.2 PREÁMBULO

El diseño y la implementación de AHPIOS están basados en la arquitectura software de ROMIO. En ROMIO, la interfaz de MPI-IO es implementada encima de la interfaz abstracta de dispositivos ADIO.

En la Figura 11 se muestra la arquitectura software de ROMIO.

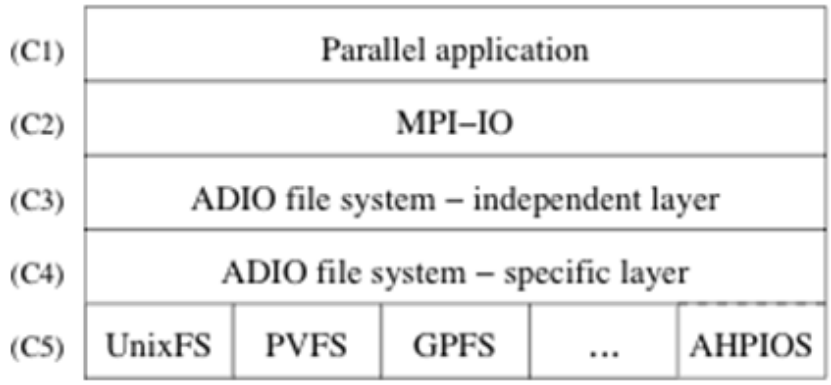


Figura 11: Arquitectura ROMIO

Como se puede observar esta divida en cinco capas. Las llamadas MPI-IO de la aplicación son traducidas en la capa (C2), en un subconjunto menor de llamadas ADIO. La capa (C3), contiene la implementación de mecanismos y optimizaciones como las vistas, acceso a ficheros no continuos y llamadas colectivas de E/S. La capa (C4) mapea a un subconjunto menor de funciones de acceso a ficheros sobre un sistema de ficheros particular. Esta capa ha tenido que ser implementada con el fin de añadir soporte ROMIO, a los nuevos sistemas de ficheros paralelos. Por último la capa (C5) consiste en las rutinas de acceso del sistema de ficheros (pudiendo ser librerías a nivel de usuario o el sistema de ficheros montado de manera local). AHPIOS ha sido integrado dentro de la pila de la arquitectura de ROMIO, mediante la implementación de las capas (C4) y (C5).

5.3 DISEÑO E IMPLEMENTACIÓN

El acceso de las aplicaciones MPI a una partición AHPIOS se hace a través de la interfaz de MPI-IO (llamándose a la implementación de la interfaz MPI-IO cliente AHPIOS). Una partición AHPIOS es gestionada por un conjunto de servidores AHPIOS, los cuales son procesos MPI. Estos procesos se ejecutan independientemente de la aplicación MPI cliente.

En la Figura 12 se muestra la arquitectura software del sistema AHPIOS, donde en la parte superior se encuentra la aplicación cliente y en la parte inferior se encuentra un conjunto de servidores AHPIOS.

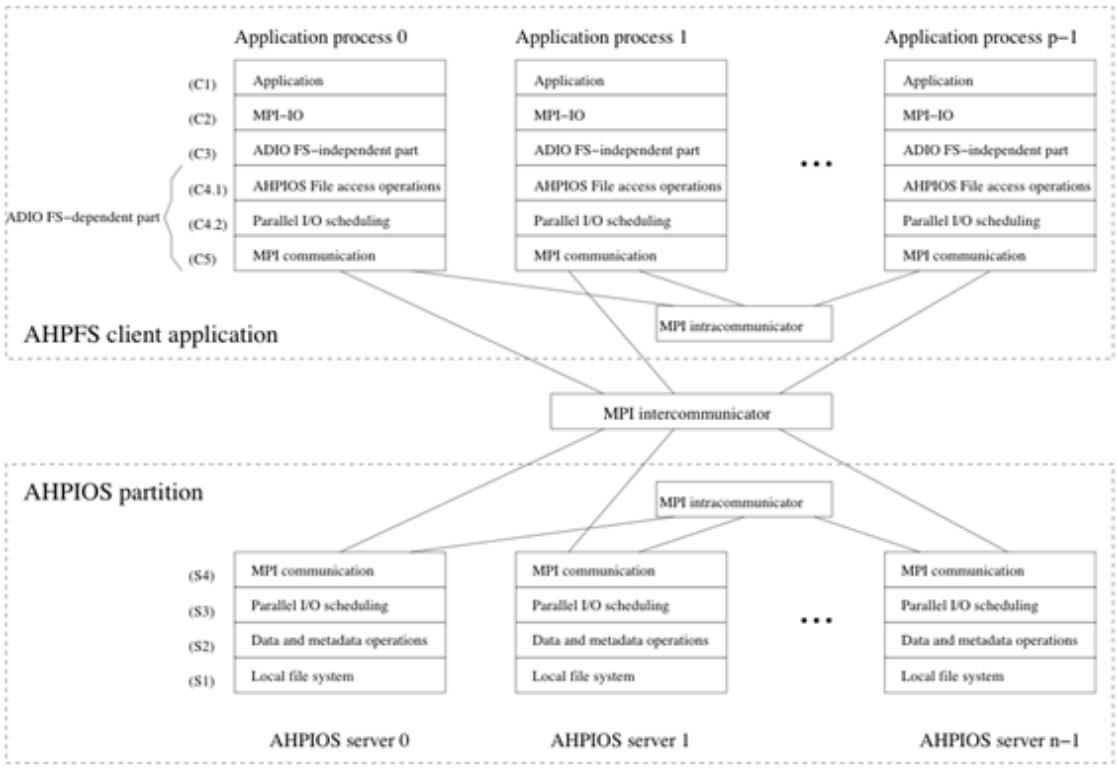


Figura 12: Arquitectura software de AHPIOS

Los servidores AHPIOS están comunicados entre sí a través de un intercomunicador de MPI (*MPI intracommunicator*). Un *MPI intracommunicator* es un mecanismo de MPI el cual permite a los miembros de un grupo de procesos comunicarse con otro grupo a partir de las rutinas de comunicación de MPI.

Los servidores AHPIOS están comunicados con la parte cliente AHPIOS a través de un intercomunicador de MPI.

El cliente AHPIOS forma parte de la interfaz de MPI-IO. Está integrado dentro de la pila de la arquitectura ROMIO en las capas (C4) y (C5). La capa (C4) puede ser dividida en dos subcapas (C4.1) y (C4.2). La subcapa (C4.1) mapea las operaciones de

fichero de ADIO sobre tareas que puedan ser realizadas por los servidores AHPIOS de manera individual. Estas tareas pueden ser relaciones de metadatos, como la creación o eliminación de un fichero u operaciones sobre ficheros. En la subcapa (C4.2) estas tareas son programadas para la transferencia por un modulo planificador de E/S paralela. La capa (C5) es la responsables de las comunicaciones con los servidores AHPIOS a través de las rutinas de comunicación de MPI.

Los servidores AHPIOS han sido diseñados como un programa MPI completo e independiente. Como se puede ver en la Figura 12, el diseño del servidor está estructurado en 4 subcapas. La comunicación con la aplicación cliente es realizada a través de rutinas MPI en la subcapa (S4). La subcapa (S3) es la responsable de la política de planificación de E/S paralela, orientada a la cooperación con los correspondientes módulos del lado del cliente. La gestión de los datos y metadatos se realiza en subcapa (S2). Finalmente, la subcapa (S1) transfiere los datos y metadatos al sistema de almacenamiento final.

Varias particiones AHPIOS con diferentes configuraciones pueden ser ejecutadas en paralelo en un *cluster* (después de registrarse en el registro global). En la Figura 13 se puede ver un ejemplo de dos aplicaciones utilizando dos particiones AHPIOS diferentes.

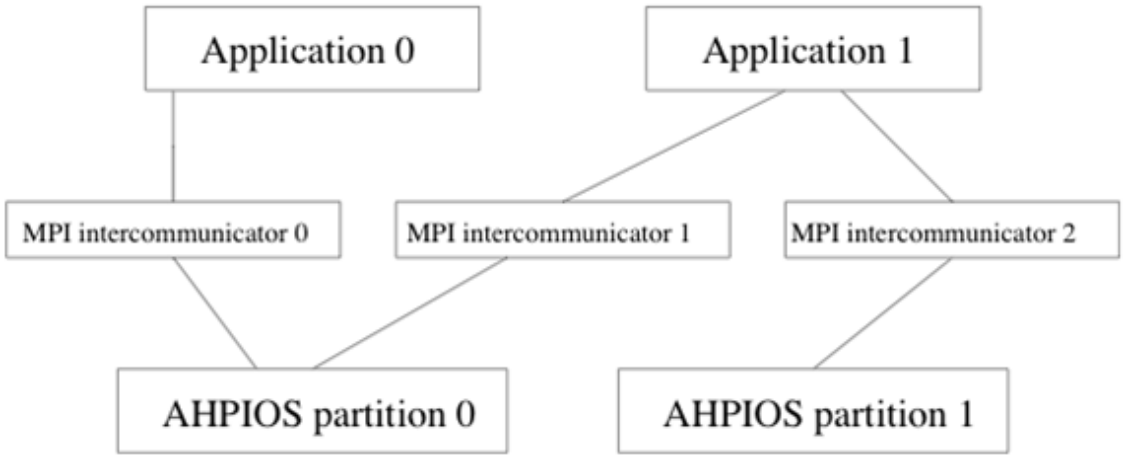


Figura 13: Diferentes particiones de AHPIOS sobre un mismo cluster.

Para cada partición montada, se crea un intercomunicador MPI dedicado, a través del cual la capa MPI-IO se comunica con los servidores AHPIOS.

Los atributos de creación de las particiones (como la cantidad de recursos, el ancho de banda y la lista de recursos usados para datos y metadatos) son almacenados en un fichero de configuración, con la misma estructura que el ejemplo siguiente:

```
# The default stripe size of the partition
stripe_size = 64k
# Number of IOS
nr_ios = 4
# Storage resources of AHPFS servers
ahpfs_server = n0:/data
ahpfs_server = n1:/data
ahpfs_server = n2:/data
ahpfs_server = n3:/data
# Path of the metadata directory
metadata = n0:/data
```

5.3.1 PARTICIONES

Las particiones en AHPIOS son elásticas: pueden aumentar o disminuir para incrementar el número de recursos de almacenamiento. Escalando hacia arriba solo implica el reinicio del sistema con un mayor número de servidores AHPIOS. En este caso, para los ficheros almacenados en el sistema antiguo con menor número de recursos, los usuarios pueden optar a conservar su estructura inicial o redistribuirla sobre los nuevos recursos de almacenamiento. Escalando hacia abajo implica una redistribución de los datos que ya no se encuentran accesibles, siendo realizado este proceso en dos pasos: primero, todos los recursos de almacenamiento antiguos son montados al iniciar el sistema con el número de servidores AHPIOS antiguos y se realiza la redistribución; segundo, la partición antigua es desmontada y la nueva montada.

5.3.2 CACHE COOPERATIVA

El acceso a los datos se realiza a través de la cooperación de las librerías de los clientes (ejecutándose en varios nodos de cómputo) y en el servidor AHPIOS. Las órdenes de transferencia de datos son controladas por un planificador de E/S paralela.

Un fichero AHPIOS debe ser particionado sobre varios servidores AHPIOS. Por defecto los ficheros son particionados sobre todos los servidores AHPIOS disponibles, pero el usuario puede controlar los parámetros de particionamiento a partir de parámetros de MPI (MPI hints).

Una partición AHPIOS se accede a través de una jerarquía de dos niveles de caches cooperativas. El usuario puede elegir si deshabilitar el primer nivel de cache cooperativa por razones de consistencia (si diferentes aplicaciones utilizan el mismo fichero, aunque los estudios han demostrado que se trata de un caso muy raro dentro de las aplicaciones científicas).

El primer nivel de cache cooperativa, es gestionado a través de la cooperación de todos los procesos de la aplicación (utilizar solo un subconjunto de procesos también es posible), quien pone en común una fracción de memoria local del nodo sobre el que se está ejecutando. De esta forma, este nivel de cache es escalable con el número de procesos. Por analogía con el método *two-phase I/O* implementado en ROMIO, estos nodos se llaman agregadores, por ellos también juntan pequeños trozos de un fichero en una página cache más grande.

El primer nivel de cache cooperativa funciona de la siguiente manera. Los bloques del fichero son mapeados mediante *round-robin* sobre todos los agregadores. La petición del fichero se dirige al responsable de los agregadores. El agregador apila junto con varias peticiones antes de acceder al siguiente nivel de cache. La comunicación con el segundo nivel de cache es realizada de manera asíncrona a partir de un hilo de E/S, el cual oculta la latencia de acceso al fichero a la aplicación.

El segundo nivel de cache cooperativa es manejado por los servidores AHPIOS. Los bloques del fichero, son mapeados a los servidores AHPIOS, con el algoritmo *round-robin*, siendo cada servidor responsable de transferir estos bloques al sistema de almacenamiento persistente. Cuando un servidor AHPIOS recibe una petición de un bloque del cual no es responsable, este servidor contesta a esta petición en cooperación con el resto de los servidores. Este enfoque es útil, al menos en dos escenarios. En primer lugar el cuello de botella de E/S relacionado con la computación puede ser disminuido de los servidores AHPIOS. En segundo lugar, en el sistema Blue Gene, un subconjunto de cómputo es asignado a un servidor de E/S, el cual es responsable de todas las peticiones sobre ficheros de este grupo, pudiéndose ahora servir al grupo en cooperación con otros servidores de E/S.

5.3.3 ACCESO A LOS DATOS

El estándar de MPI-IO define dos grandes grupos de operaciones de E/S: colectivas e independientes. En el caso de AHPIOS, las operaciones independientes son realizadas de la misma manera en el *server-directed I/O* explicado más adelantes en este mismo apartado, por lo que nos concentraremos en las operaciones colectivas.

Las operaciones colectivas son apropiadas para los trabajos paralelos debido a las siguientes cuatro características comunes en la aplicaciones de datos intensos de tipo científica. Primero, es frecuente que todos los nodos de cómputo realicen el acceso al mismo fichero. Segundo, cada nodo accede al fichero de manera no contigua y con una granularidad muy pequeña. Tercero, existe una gran degradación de la localización de espacio (cuando un nodo accede a algunas regiones del fichero, los otros nodos tienden a acceder a los datos vecinos). Cuarto, el acceso de escritura de datos es mayoritariamente no superpuesto.

Como se comentó anteriormente, la distribución de ROMIO contiene la implementación *two-phase I/O*, donde los buffers colectivos residen en los agregadores (los cuales son una parte de los nodos de cómputo). El diseño de AHPIOS incluye dos métodos colectivos de E/S, donde ambos están basados en vistas: *view-based client-*

directed y *view-based server-directed*. A continuación se muestra una comparación entre *two-phase I/O*, *client-directed* y *server-directed*.

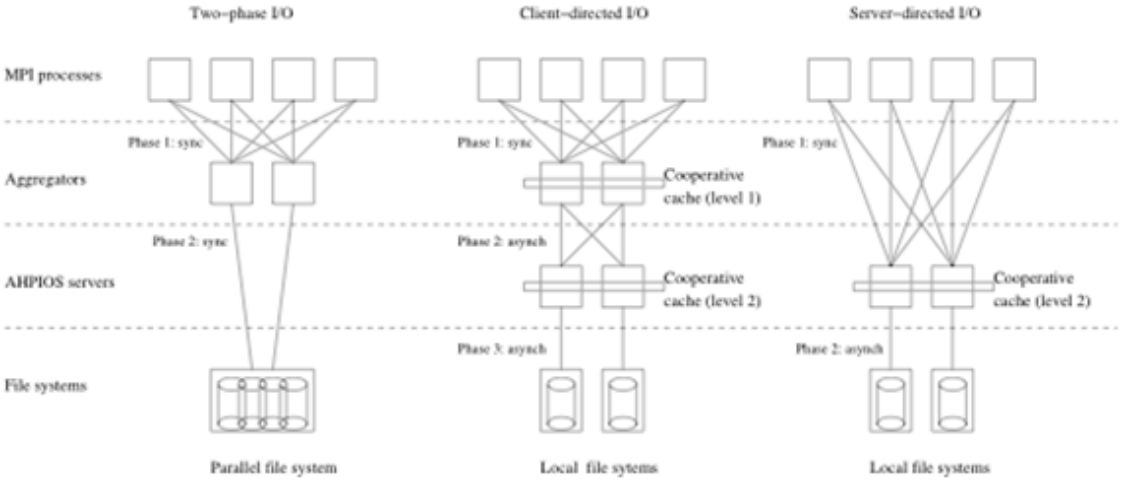


Figura 14: Comparativa funcionamiento métodos de E/S.

Asumiendo que la vista ya ha sido declarada, la descripción es almacenada en el proceso cliente para el método *two-phase I/O*. Para client-directed *I/O* la vista es enviada a los agregadores, donde es almacenada para su uso futuro. Para *server-directed I/O* la vista es enviada a los servidores de AHPIOS, donde también será almacenada para su uso futuro.

Tanto para *two-phase I/O* como para *client-directed I/O,* el agregador representa un subconjunto del proceso MPI y son usados para agrupar pequeñas peticiones en una grande. Por defecto, todos los procesos MPI actúan como agregadores. Sin embargo el usuario puede configurar el número de agregadores a partir de un argumento configurable de MPI.

Client-directed I/O consiste en tres fases. Cuando el client-directed *I/O* empieza, las vistas ya han sido almacenadas en los agregadores como una declaración. La primera fase es síncrona y consiste en repartir los datos entre los procesos MPI y los agregadores (pequeñas regiones del fichero son reunidos en el agregador para la escritura y son separados de los agregadores para la lectura). Las pequeñas regiones del fichero son

transferidas de manera contigua entre cada par de procesos MPI y agregadores, y son agrupados o separados según la declaración de la vista almacenada anteriormente. En la segunda fase, los datos son enviados de manera asíncrona desde el primer nivel de cache cooperativa a los servidores AHPIOS. Solo bloques de fichero completos son transferidos entre estos dos niveles de cache. Finalmente en la tercera fase, los datos son almacenados asíncronamente también desde la cache de los servidores AHPIOS al sistema de almacenamiento final.

Server-directed I/O consiste en dos fases: Cuando el *Server-directed I/O* comienza, las vistas han sido también almacenadas en los servidores AHPIOS. La primera fase, es síncrona similiar que con el *client-directed I/O,* exceptuando el hecho de que los datos son repartidos entre los procesos MPI y los servidores AHPIOS. La segunda fase es asíncrona y es igual que la tercera fase del *client-directed I/O.*

Se puede notar que existen dos grandes diferencias entre *client-directed I/O* y *server-directed I/O*: el lugar donde las vistas son almacenadas y el nivel intermedio de cache para el *client-directed I/O.* En consecuencia, pequeñas peticiones son unidas en grandes por los agregadores del *client-directed I/O* y en los servidores AHPIOS para *server-directed I/O*.

La tabla que se muestra a continuación compara el método *two-phase I/O* con los métodos *client-directed I/O* y *server-directed I/O*

Operación	Two-phase I/O	Client-directed	Server-directed
Declaración de vista	Almacena vista en cliente	Envía vista a los agregadores AHPIOS	Envía vista a los servidores AHPIOS
Acceso fichero (metadatos)	Genera la posición del último acceso de las vistas y se lo manda a los agregadores.	n/a	n/a
Acceso fichero (cliente)	No contiguo	Contiguo.	Contiguo.
Acceso fichero (agregador)	No contiguo	No Contiguo.	n/a.
Acceso fichero (servidor)	Contiguo	Contiguo	No contiguo
Cache de fichero	No.	Primer nivel de cache cooperativa.	Segundo nivel de cache cooperativa.
Transferencia datos desde agregadores hacia servidores	Sincrona.	Asincrona.	n/a.
Transferencia datos desde servidores hacia almacenamiento	n/a.	Asincrona.	Asincrona.
Cerrar fichero	n/a.	Tras verificar que todos los agregadores AHPIOS y servidores han puesto todos los datos.	Tras verificar que todos servidores han puesto todos los datos.

Tabla 2: Comparativa métodos de acceso.

A diferencia de *two-phase I/O*, para *client-directed* y *server-directed I/O*, las vistas, representadas como tipo de datos, no son almacenadas en la aplicación del cliente, pero son decodificadas por la capa MPI-IO, serializadas y enviadas ya sea para agregadores de AHPIOS o servidores AHPIOS. Tras la recepción de la vista, los agregadores o los servidores deserializan y reconstruyen el tipo de datos original de la vista. La ventaja de esta solución es que los metadatos no son necesarios enviarlos por

la red en el tiempo de acceso, porque la vista representa el acceso a ficheros almacenados de manera remota. Para *two-phase I/O* el patron de acceso generado por la vista, tiene que ser enviado como una lista de tuplas (file offset, length). En el caso de *client-directed* y *server-directed I/O* los datos pueden ser enviados contiguous entre el cliente y el agregador/servidor. En *two-phase I/O*, no está cacheado en agregadores, pero solo de manera temporal es agrupado y enviado de manera síncrona al sistema de ficheros. En *client-directed I/O* los datos del fichero son cacheados en dos niveles de jerarquía de caches cooperativas son enviados de manera asíncrona al sistema de almacenamiento final. En *server-directed I/O,* los datos son cacheados en el segundo nivel de la jerarquía de cache cooperative y son enviados de manera asíncrona al almacenamiento. Las transferencias asíncronas permiten una superposición de cómputo de manera transparente, E/S relacionadas con la comunicación y el acceso al almacenamiento.

5.3.4 COHERENCIA CACHÉ Y CONSISTENCIA DEL FICHERO

La coherencia cache está orientada a que ambos niveles de cache no almacenen más de una copia de los bloques de datos en cada nivel. Esta decisión es tomada debido a la frecuencia de los patrones de acceso en las aplicaciones paralelas: procesos independientes no sobrescriben regiones del fichero y existe una alta localización espacial entre procesos. El grano de acceso más pequeño que un bloque, ocurre solo en el primer nivel de cache, por ejemplo cuando se reparte los datos para operaciones colectivas. Los datos son transferidos entre ambos niveles cache siempre con la granularidad de un bloque. Para escribir en un bloque del fichero menor que su tamaño en el primer o segundo nivel de cache, se necesita una operación de lectura-modificación-escritura, donde la operación final de escritura es realizada de forma asíncrona. Para *client-directed I/O*, la modificación de un bloque de fichero, siempre se realiza por un agregador del primer nivel de cache. *Server-directed I/O,* no usa el primer nivel de cache, por lo tanto la coherencia está orientada solo a los servidores, mediante el alojamiento de una copia del bloque del fichero. Existe un disparador en el *Server-directed I/O* que provoca la negación de acceso a los bloques de fichero de la cache de

primer nivel desde el servidor, antes de que este haya terminado de realizar sus operaciones.

De acuerdo al estándar de MPI, MPI proporciona tres niveles de consistencia: consistencia secuencial a través de todos los accesos (usando un descriptor único del fichero), a través de todos los accesos (usando descriptores de ficheros creados tras la operación de apertura de modo atómico), y por último mediante la consistencia definida por el usuario.

AHPIOS proporciona una implementación parcial de la consistencia de MPI. El acceso colectivo desde una aplicación es secuencialmente consistente, es decir, los datos son llevados al almacenamiento final desde los dos niveles de cache ya sea llamando a MPI_FILE_SYNC o cerrando el fichero. El modo atómico para la independencia de acceso a los datos no se ha implementado, dado que la superposición de accesos no es frecuente en las aplicaciones paralelas y puede ser aplicada mediante consistencia semántica por el usuario, mediante la función MPI_FILE_SYNC, como se describe en el estándar de MPI.

5.3.5 METADATOS

En AHPIOS existe dos niveles de manejo de metadatos: nivel global de manejo de metadatos de las particiones AHPIOS y nivel local de cada partición.

- **NIVEL GLOBAL:** En AHPIOS no existe un servicio que realice la gestión de los metadatos globales. Los metadatos globales son mínimos y contienen solo información particular de las particiones del sistema de ficheros. Esta información es almacenada en un fichero compartido por todas las particiones llamado *registry*. Cada conjunto de servidores AHPIOS, gestiona el acceso a este fichero de manera atómica tanto para lectura como para modificarlo. Este acceso no debe causar un cuello de botella debido a que el *registry* solo es accedido cuando, se monta o se desmonta una partición. Siendo todas estas operaciones poco frecuentes.

El fichero *registry* almacena la estructura y la configuración dinámica de los parámetros de las particiones AHPIOS. Los datos estructurales son el nombre de la partición, los recursos de almacenamiento asignados a los servidores AHPIOS, el número de servidores, el tamaño por defecto de bloque y localización del fichero de metadatos. Los parámetros dinámicos incluyen el tamaño del buffer de red, el planificador de la política de E/S, el tamaño del buffer cache del servidor AHPIOS, etc. Los parámetros dinámicos pueden ser cambiados por el usuario cada vez que se activa la partición.

- **NIVEL LOCAL**: Por cada partición, uno de los servidores AHPIOS, juega el rol de gestor de metadatos. Este servidor maneja un espacio de nombres local y una lista de inodos, almacena y recupera el archivo de metadatos, y finalmente actualiza los metadatos en coordinación con el resto de los servidores. El nombre global de un fichero viene dado por lo tanto, por la adición de la ruta local del fichero y el nombre global único de la partición. El espacio de nombres local puede ser simplemente un directorio del espacio de nombres del sistema de ficheros del nodo de cómputo donde el servidor AHPIOS se está ejecutando. El almacenamiento típico de inodos en el fichero de metadatos viene definido por el valor del tamaño de bloque y el número de sectores. Por defecto, el número de sectores es el mismo que el número de servidores AHPIOS, el usuario puede modificar este valor a partir de un argumento.

6. ETAPAS DE LA MIGRACIÓN

Migrar un sistema desarrollado, no es siempre una tarea fácil. De hecho la mayoría de las veces es una tarea ardua y dura. Esto se debe a que muchos desarrollos se basan en código propietario o existe incompatibilidades entres los distintos sistemas operativos.

Para realizar la migración de AHPIOS hacia plataformas Windows®, se ha tenido que realizar, aparte de las modificaciones necesarias para la compatibilidad de funciones y tipos de datos, un estudio sobre la viabilidad de la migración.

A lo largo del siguiente capítulo, se comenta la metodología seguida para la realización de la migración del sistema de E/S AHPIOS, así como las dificultades encontradas y las medidas tomadas para su correcta incorporación. Para la ejecución de la migración se han seguido los siguientes pasos:

- **Estudio posibilidad de migración**: comprobar si es posible integrar MPI dentro del entorno de Windows®, estudiando si el sistema operativo de Microsoft ofrece la posibilidad de ejecutar este tipo de aplicaciones.

- **Integración MPI en entornos Windows®**: realizar la instalación de la distribución MPICH sobre el sistema operativo Windows® y realizar las pruebas correspondientes para verificar el correcto funcionamiento bajo dicho sistema operativo.

- **Incorporación de las librerías de AHPIOS**: añadir las librerías de funcionamiento del sistema de E/S paralela AHPIOS, dentro del proyecto de MPI, realizando las correspondientes modificaciones de desarrollo para adecuar la implementación existente para el sistema operativo Linux.

- **Incorporación del servidor AHPIOS**: integrar el servidor de E/S de AHPIOS dentro de la solución.

- **Instalación en las aulas docentes**: instalar todo el sistema desarrollado en el laboratorio, junto con las restricciones que estas conllevan, generalmente restricciones de seguridad y permisos de usuarios.

6.1 ESTUDIO DE VIABILIDAD DE LA MIGRACIÓN

Lo primero que se ha realizado es el estudio de la posibilidad de incorporar MPI dentro de un entorno Windows®. Para ello se han buscado referencias sobre la posibilidad de esta instalación.

Microsoft HPC Server 2008 es una clara referencia de la posibilidad de incorporación de MPI sobre sistemas Windows®, debido a que incorpora una implementación de MPI (*Message Passing Interface*) basada en MPICH2 (desarrollada por el MCS en *Argonne National Laboratory)*.

MPICH2 se trata de una implementación libre del estándar MPI, esta distribución aporta documentación sobre la posibilidad de la instalación de MPI sobre cualquier entorno Windows®, además de ofrecer dos métodos de instalación del sistema de paso de mensajes, código fuente o binario. Durante el resto del capítulo nos centraremos en la instalación de código fuente, que nos posibilita la incorporación de las librerías del sistema de E/S paralela AHPIOS.

6.2 INTEGRACIÓN MPI EN ENTORNOS WINDOWS®

En este aparatado se detalla el proceso de integración de la instalación básica de MPI en un entorno Windows®. Para ello como se ha comentado anteriormente, la instalación se realizará a partir de la distribución MPICH (en su versión de código fuente).

Lo primero que se ha realizado, es la obtención de información de cómo instalar el estándar del sistema de paso de mensajes MPI, para ello se hace necesario la compilación del código fuente, a partir de la solución proporcionada realizando los siguientes pasos:

- Ejecutar el script winconfigure.wsf con la opción de --remove-fortran.
- Compilación de toda la solución con la configuración ch3sockRelease.
- Compilación del proyecto mpich2s con la configuración ch3sockRelease.
- Compilación de toda la solución con la configuración Release.

Tras la realización de los pasos anteriores, se consigue la generación de las librerías necesarias para la compilación y ejecución de las aplicaciones basadas en MPI, así como los ejecutables.

Una vez generado todo lo necesario para la ejecución de aplicaciones, se compila un ejemplo básico llamado "Hola Mundo", el cual nos indica el identificador del nodo y muestra un mensaje de "Hola Mundo" por pantalla. Durante la compilación de esta prueba básica, surgió un problema de imposibilidad de compilación debido a que no se encontraban las librerías de MPI. Por ello, surgió la necesidad de crear un vínculo de las librerías generadas con el proyecto que contiene el código fuente de la aplicación MPI que se quiere compilar.

A la hora de ejecutar la aplicación MPI, se hace necesario que se esté ejecutando un servicio en el sistema operativo. Este servicio es conocido como *smpd*, el cual hay que instalar en cada nodo de cómputo que se quiera usar en la aplicación MPI. Para su posible funcionamiento, la instalación consiste en ejecutar el siguiente comando en la consola de comandos del sistema operativo Windows®:

```
smpd –install
```

El ejecutable smpd.exe es generado durante la compilación del código fuente. El servicio *smpd* permite ejecutar aplicaciones bajo múltiples credenciales de usuario, permitiendo logearse en los nodos de cómputo como usuario y ejecutar procesos en ellos.

Tras la comprobación del correcto funcionamiento del sistema de paso de mensajes MPI a partir de la aplicación creada, se realiza una serie de pruebas para

comprobar su completo funcionamiento bajo el sistema operativo Windows®. Para ello se genera una serie de programas basado en MPI, que nos permiten probar las siguientes funcionalidades necesarias de MPI:

- Llamadas MPI.
- Llamadas colectivas de MPI.
- Spawn.
- Llamadas de E/S.

6.3 INCORPORACIÓN DE LAS LIBRERIAS DE AHPIOS

Para la integración de las librerías AHPIOS dentro de ROMIO, ha sido necesaria la codificación de las librerías que se encuentran en las capas (C4) y (C5) de la arquitectura de ROMIO, expuesta en la Figura 15:

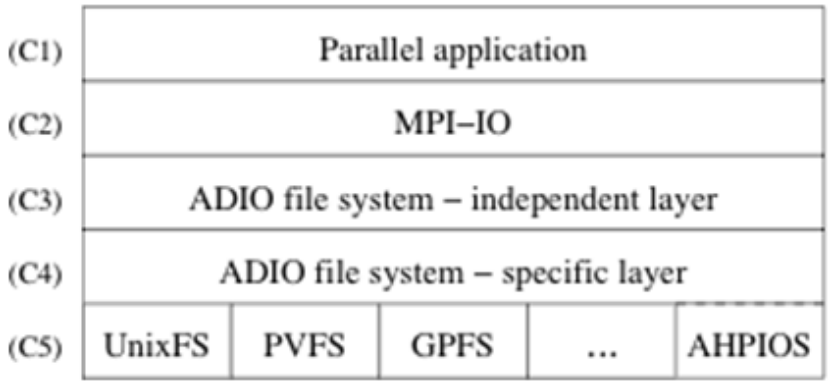


Figura 15: Arquitectura software de una aplicación paralela en MPI y ROMIO

En la capa (C4) se integran dos tipos de operaciones, las operaciones de ficheros de ADIO (como puede ser creación o eliminación de un fichero) y las operaciones para la transferencia, siendo este el planificador de E/S paralela. En la capa (C5) se encuentran las rutinas de comunicación entre los clientes y servidores AHPIOS, las cuales usan el estándar MPI.

A continuación se explican los pasos realizados para la integración de las librerías dentro del proyecto de .NET, así como los problemas que han surgido durante el proceso de incorporación y las soluciones tomadas para su correcto acoplamiento.

Lo primero que se ha realizado ha sido añadir los ficheros de código fuente de las librerías dentro del proyecto.

Tras añadir las librerías dentro del proyecto, se hace necesario la modificación del mismo, para que reconozca los ficheros nuevos que se han incluido, para ello se ha hecho necesario primero la modificación del script de instalación ·wincofigure.wsf", añadiendo a la información que contiene, la definición de AHPIOS. Esta definición consiste en una variable global que nos permita comprobar si está integrado AHPIOS o no. En segundo lugar, se ha tenido que añadir las librerías al proyecto, esto se puede realizar de dos maneras:

- Modificar los ficheros "mpich2.vcproj" y "mpich2s.vcproj" con un editor de texto plano y añadir las librerías nuevas dentro de la zona del fichero donde se encuentran definidas las librerías actuales incorporadas en el proyecto. La zona del fichero donde se encuentran definidas las librerías, se puede identificar por tener la siguiente nomenclatura XML:

```
<File
      RelativePath="ruta_libreria"
      >
</File>
```

Debiéndose añadir una entrada XML, con la misma estructura que la arriba definida, por cada librería que se tiene que añadir. Las librerías nuevas que añadir son las que se encuentran dentro del sub-directorio "common" dentro de directorio ADIO, y las que se encuentran dentro del sub-directorio "ad_ahpfs".

- Abrir la solución y añadir las librerías en ambos proyectos, para ello se selecciona uno de los proyectos dentro del panel de soluciones y se despliega el directorio de su contenido, una vez realizado aparecen tres carpetas, dentro de la carpeta "Header Files" se cliquea con el botón derecho del ratón dando a agregar elemento existente, tras esto aparece un explorador de Windows®, y se selecciona los ficheros de cabecera (.h) que se encuentran dentro del subdirectorio "ad_ahpfs", una vez realizado se procede a ejecutar el mismo mecanismo pero esta vez sobre la carpeta "Source Files", debiéndose añadir ahora los fichero de código fuente (.c) que se encuentran en la carpeta "common" y en la carpeta "ad_ahpfs".

Una vez se ha finalizado el proceso de inclusión de las librerías se procede a realizar la compilación del proyecto para crear las librerías dinámicas (.dll), encontrándose aquí el primer problema de compatibilidad, debido a que muchos ficheros de código fuente añadido tenían dependencias con librerías exclusivas para Linux, estas librerías estaban relacionadas con las llamadas al sistema para la creación de procesos, pudiéndose eliminar la inclusión en las cabeceras de los ficheros debido a que no se hacía necesario su uso dentro del sistema de E/S paralela.

Tras la solución del problema anterior, surgió otro problema y es la incompatibilidad de las llamadas POSIX, para el manejo de ficheros en Windows®, debido a que no reconocía las funciones. Para poder arreglar este inconveniente, se ha realizado una traducción de las llamadas POSIX para el manejo de ficheros a las llamadas proporcionadas por el proyecto dentro del sistema de fichero NTFS, se ha tomado esta medida, debido a que en la actualidad la plataforma Windows® trabaja sobre este sistema de ficheros.

Además también surgió la incompatibilidad de otras llamadas POSIX como son para las operaciones de mapeo en memoria, la cuales se usan como un método de ejecución en AHPIOS que permite volcar un fichero a memoria y trabajar con él, al no encontrase compatibilidades en las funciones con las que ofrece la plataforma .Net, se ha optado por la supresión de este método de funcionamiento en AHPIOS.

Otro problema que ha surgido durante la migración, ha sido el no reconocimiento por parte de la plataforma .NET de las Macros de programación. Las Macros son una sección de código definida para que se compile si reconoce una variable de entorno y que la ignore en caso contrario. Solucionándose este problema, mediante la eliminación de las Macros.

Además de los problemas con las Macros, existían problemas de no reconocimiento con algunos tipos de datos y llamadas a funciones, realizándose la sustitución del tipo de datos por otro homólogo pero que reconociera la plataforma .Net y cambiando las llamadas a funciones, por otra que hiciera la misma funcionalidad.

Por último, una vez solucionados todos los problemas de compilación generados durante la migración del proyecto, se obtuvo el último gran problema que venía generado por errores de linkeo. Estos errores, se producen, cuando una librería que tiene dependencia con otra, debido a que utiliza sus funciones, no encuentra dichas funciones, interpretando que no están definidas. El problema no era que generase dicho error, sino el porqué, debido a que las librerías por las que se producía el error de linkeo, sí que se encontraban dentro del proyecto y con las dependencias de uso bien definidas. Al principio se estuvo revisando todas las dependencias y ajustando para minimizar el número de errores, hasta que hubo un momento en el que no se podía arreglar nada más, llegándose a un callejón sin salida. En esta situación se procedió a realizar un cambio de perspectiva, se cogió una copia de la distribución de MPICH limpia, es decir, una copia original de las proporcionadas por MPICH. Una vez con la copia limpia, se procede a incorporar poco a poco las librerías nuevas al proyecto, siguiendo los pasos comentados con anterioridad, tras cada fase de incorporación de librerías, se realiza una

compilación del proyecto para comprobar que todo es exitoso y continuar añadiendo otro conjunto de librerías. Realizando este paso conseguimos incorporar todas las librerías al proyecto reduciendo en un 80% los errores de linkeo que se tenían, aunque aun exista unos pocos errores de linkeo, estos errores venían dados por la librería de MPI, ya que no reconocía bien las llamadas a algunas de sus funciones, esto se solucionó mediante la utilización de la interfaz abstracta de las llamadas a dichas funciones, es decir, realizando las llamadas a un nivel superior.

Tras la generación de las librerías, incluyendo la funcionalidad de AHPIOS, se procede a realizar una serie de pruebas para comprobar que todo sigue funcionando correctamente bajo el sistema operativo Windows®. Para ello se genera una serie de programas MPI, que permite probar las siguientes funcionalidades:

- Llamadas MPI.
- Llamadas colectivas de MPI.
- Spawn.
- Llamadas de E/S.

La funcionalidad de AHPIOS, tanto a nivel de cliente como del servidor, se prueba en la última fase de la migración, debido a que sin tener las dos partes completamente integradas, resulta imposible probar el funcionamiento.

6.4 INCORPORACIÓN DEL SERVIDOR AHPIOS

En esta última fase de migración de AHPIOS a Windows®, se procede a generar la aplicación que ejecuta el código del servidor AHPIOS, para que posteriormente los nodos del cluster que ejecuten este código puedan interactuar como servidor AHPIOS en el cluster.

Lo primero que se ha tenido que modificar han sido las cabeceras, debido a que hacían referencia a librerías de manejo de memoria bajo POSIX, no siendo compatible la plataforma .Net con ellas. Esta supresión de la cabecera, obliga a eliminar las zonas de código que hacían referencia al manejo de memoria, siendo estas funciones las que realizan la utilización del fichero mapeado en memoria y por consiguiente se elimina dicha funcionalidad.

Además, surgió el problema de la incompatibilidad de las llamadas POSIX, para el manejo de ficheros en Windows®, debido a que no reconocía las funciones. Para poder arreglar este inconveniente, se ha realizado una traducción de las llamadas POSIX para el manejo de ficheros a las llamadas proporcionadas por el proyecto dentro del sistema de fichero NTFS, se ha tomado esta medida, debido a que en la actualidad la plataforma Windows® trabaja sobre este sistema de ficheros.

Otro problema que ha surgido es el no reconocimiento de algunos tipos de datos, debiéndose hacer la sustitución del tipo de datos por otro homólogo pero que reconociera la plataforma .Net y también se ha tenido que realizar algunos casting para el manejo correcto de los datos. Además del problema con los tipos de datos surge también el inconveniente del no reconocimiento de algunas funciones, debiéndose sustituir estas por otra homóloga que haga la misma funcionalidad.

Por último surgió el problema de que no reconocía las llamadas a algunas funciones declaradas dentro del propio cuerpo del código del servidor, para solucionar este problema se optó por poner la declaración de las funciones al principio para su correcto reconocimiento.

Tras la realización de los cambios necesarios en el código del servidor de AHPIOS. Se procede a realizar la generación del mismo, realizándose varios intentos de generación distinta hasta que se consiguió dar con el valido:

- Primero se realizó la creación de un proyecto externo, a la solución de MPICH, en la cual se añadió el fichero de código fuente del servidor, y se

configuró todas las dependencias del proyecto, para que enlazara con las librerías que se generó durante la fase de incorporación de las librerías de AHPIOS. El problema de esta medida fue se tenían errores de linkeo al igual que pasaba en la fase anterior cuando se estaba añadiendo las librerías de AHPIOS al proyecto. Se intentó la modificación de la forma de compilar las librerías y cambiar las dependencias del proyecto, pero no se consiguió llegar a una solución válida para generar el servidor de AHPIOS.

- Al comprobar que como proyecto externo no se consigue realizar la generación del servidor, se procede a generar un proyecto interno dentro de la solución del proyecto, donde se añade el código fuente del servidor y se ajustan las dependencias con los otros proyectos que incorporan la distribución de MPICH. Esta solución tomada también reporto los errores de linkeo al igual que nos pasó con la medida tomada con anterioridad, intentando al igual que hicimos con el proyecto externo solucionarlos sin llegar a una solución validad.

- Como al crear proyectos nuevos, no se obtenía unos resultados buenos, se procede a añadir el código fuente del servidor de AHPIOS, dentro de uno de los proyectos, donde se añadieron las librerías de AHPIOS, que ya se incluyen dentro de la distribución, haciéndose necesaria la modificación de las características del proyecto, para que según el método de compilación que se usara generara la librería o el ejecutable del servidor de AHPIOS, esta solución tomada conseguimos que no produjera errores de linkeo, pero no se conseguía que generara el código ejecutable del servidor de AHPIOS, lo cual sirvió para tomar la última y correcta decisión para generar el servidor.

- La última medida tomada y la que proporcionó la solución que se estaba buscando fue la generación de un proyecto interno dentro de la solución de MPICH y añadir a este proyecto todos los ficheros que se incluyen dentro de la distribución, es decir, añadir los ficheros que forman parte del

proyecto “mpich2”, una vez realizado esto, se hace necesario ajustar las dependencias y configurar el proyecto para que genere un ejecutable.

Tras conseguir generar el ejecutable del servidor AHPIOS, se procede a realizar las pruebas de funcionamiento para comprobar su correcto acoplamiento al sistema Windows®, para ello se ejecutó una prueba básica que consiste en leer un fichero y que cada proceso que interviene en la ejecución escriba un fragmento, surgiendo un problema al realizar la ejecución, tras localizar el error se comprueba que viene dado por las operaciones sobre ficheros debido a que no se realiza de manera correcta las operaciones de apertura, cierre, lectura, etc. sobre los ficheros. Para solucionar este problema se ha realizado un mapeo en el lado del servidor, el cual consiste en realizar las operaciones sobre ficheros implementadas en ese mismo nivel.

6.5 PLAN DE VERIFICACIÓN

A continuación se detallan las pruebas que se ha realizado durante toda la fase de migración, que han servido para verificar el correcto funcionamiento del sistema.

- Se ha probado a ejecutar la llamada MPI_Comm_spawn (esta función resulta muy útil en el sistema de E/S paralela que se ha migrado, al permitir crear un conjunto de procesos hijos de forma dinámica, a partir de un conjunto de procesos padres). Para la prueba de esta funcionalidad se ha realizado la ejecución con distinto número de procesos y máquinas, trazando el correcto funcionamiento de la función.

- Trazado de todas las llamadas de E/S realizadas por los clientes y servidores, para comprobar su correcto funcionamiento, al visualizar el correcto intercambio de mensaje de MPI realizado entre ambos.

- Comprobado mediante trazas por pantalla, que el offset correspondiente a cada proceso sea el correcto, es decir, cada proceso que forma parte de la ejecución coge su trozo de fichero correspondiente para realizar su cálculo.

- Se ha verificado, que la realización de la ejecución ha sido correcta, mediante la comprobación del fichero generado, verificando que el contenido escrito dentro del mismo se corresponde y no existan huecos sin datos. Esta comprobación se ha realizado mediante la comparación del fichero generado por el sistema de E/S paralelo y el mismo fichero generado al ejecutar la aplicación en local.

6.6 INSTALACIÓN EN LAS AULAS DE INFORMÁTICA

Por último, tras conseguir el correcto funcionamiento del sistema de E/S AHPIOS migrado a un entorno Windows®, a nivel local y en un cluster virtual se procede a realizar la instalación del mismo en las aulas docentes de la universidad, la cual tiene la siguiente arquitectura:

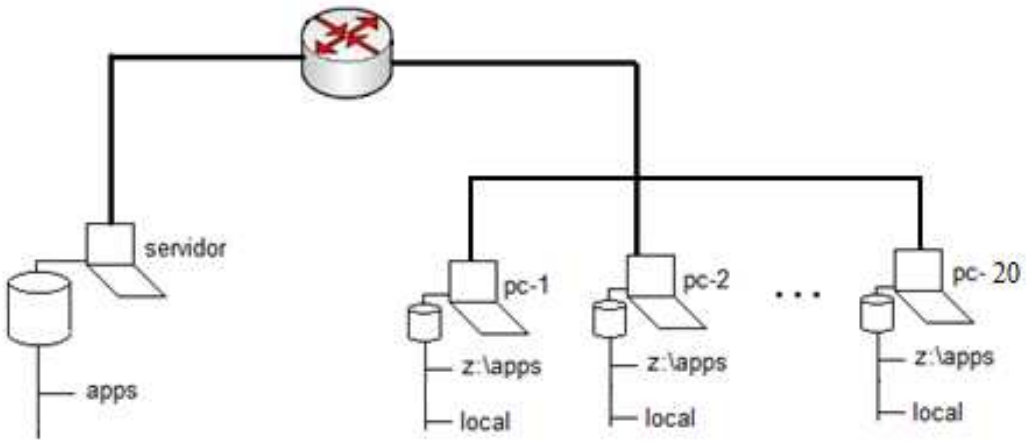


Figura 16: Arquitectura aulas docentes

El primer problema que ha surgido ha sido la instalación del servicio smpd, en los nodos del cluster, debido a la falta de permisos de administrador en dicho sistema. Teniendo que esperar a la instalación de dicho servicio por parte de administradores del sistema para poder continuar con el proceso de prueba en el aula docente.

Tras la instalación del servicio, se procede a ejecutar en una máquina una aplicación MPI. A la hora de ejecutar sobre los directorios locales del nodo, no existe ningún problema, surgiendo este en el momento de ejecutar la aplicación sobre el directorio compartido (por todos los nodos del cluster) por Samba. El problema viene dado al no reconocer la ruta de la unidad montada por red, con la ruta proporcionada por el sistema operativo, es decir, z:\directorio\app. Para solventar dicho problema se hace necesario modificar la forma de proporcionar el directorio de donde se encuentra alojados los ficheros de la aplicación, escribiéndose en formato red, siendo un ejemplo de este formato:

\\\\recurso-red\\directorio1\\directorio2\\fichero

La duplicación de la barra se hace necesaria para la correcta representación en memoria, representando dos barras como una barra.

6.7 RESUMEN

A continuación se presenta un breve resumen de las modificaciones que se han hecho necesarias realizar para la migración del sistema de E/S paralela en Linux a la plataforma Windows®:

- Configuración del fichero de generación del proyecto "winconfigure.wsf" para la plataforma Windows.
- Modificación del proyecto de MPICH, para el reconocimiento de las nuevas librerías.
- Eliminación includes propios del sistema Linux.
- Traducción de las llamadas POSIX, para el tratamiento de ficheros, por las correspondientes para el sistema de ficheros NTFS.
- Eliminación macros incompatibles
- Traducción funciones que no reconoce .Net por homólogas.
- Sustitución algunos tipos de datos, por otros que reconociera la plataforma.
- Realización de *casting* sobre algunos tipos de datos para su correcto tratamiento.
- Supresión de la funcionalidad de mapeo del fichero en memoria por incompatibilidad con las librerías.
- Configuración de las dependencias del proyecto, para la correcta generación de las librerías dinámicas (dll).
- Instalación del sistema en el laboratorio.

7. EVALUACIÓN

Durante el siguiente capítulo se expone como se ha llevado a cabo la evaluación del sistema migrado. Para ello se ha realizado una serie de pruebas de rendimiento para evaluar el sistema migrado en distintos escenarios.

Para la ejecución de la evaluación del sistema, se ha diseñado un plan de pruebas, consistente en la ejecución de una serie de sistemas de evaluación, los cuales nos proporcionan unos resultados de cómo ha sido realizada la tarea. Esta especificación de plan de pruebas viene detalla en el siguiente apartado, en donde se explica cuales van a ser los sistemas de evaluación y que resultado arrojan estos.

La ejecución de la pruebas se ha realizado en un laboratorio de ordenadores, pertenecientes a la Universidad Carlos III, el cual está formado por 20 equipos: Cada nodo del laboratorio tiene las siguientes características hardware y software:

- Windows XP service pack 3
- AMD Athlon(tm) 64 X2 Dual Core Processor 4200+
- Cache nivel L1 512 Kbytes
- Memoria RAM 2 GB

Todos los nodos se encuentran comunicados, a través de una red Ethernet de 100 MB/s.

7.1 PLAN DE EVALUACIÓN

A continuación, se expondrá los resultados que se han obtenido de la ejecución de los sistemas de evaluación.

Con el plan de evaluación, que se expone a continuación, se estudia el rendimiento que se tiene en la actualidad de los sistemas de ficheros ofrecidos por los principales sistemas operativos (Windows® y Linux), así como una comparación entre el sistema de E/S desarrollado y el ofrecido por los distintos sistemas operativos. Por último, se compara el rendimiento del sistema de E/S paralela AHPIOS en los entornos Windows® y Linux.

Para la realización de todas las evaluaciones, se ha realizado una serie de test con las herramientas comentadas en el Capítulo 4 (Herramientas de evaluación de E/S) del presente documento.

Para esta evaluación se ha utilizado la herramienta de evaluación SimParI/O. Para cada una de las evaluaciones se ejecutó 10 veces para llamada, mostrando en las gráficas el valor medio. El fichero generado por cada ejecución es igual a:

```
Tam_fichero = Tam_Bloque * Num_procesos * num_iteraciones
```

Por ejemplo, el mayor fichero generado por el benchamrk será de 812 Mbytes. El tamaño de los paquetes usados por AHPIOS es de 64 Kbytes. Un ejemplo del fichero de configuración de AHPIOS usando durante la evaluación puede ser el siguiente:

```
# This is the configuration file

# Number of IOS
nr_ios = 4

# The blocksize of the filesystem
block_size = 64k

# What is this?
net_buf_size = 64k

# Sheduling policy (0..2)
# 0: zero 1: random 2: hash
scheduling = 2

# File access at  of IOSERVER
# 0: map,  1: data sieving,  2: fs syscalls
file_access_type = 2

ioserver_exec_path=\\\\lab.inf.uc3m.es\\users\\docencia\\si\
\APP\\ioserver.exe
ioserver_data_path=C:/Documents and Settings/si/Escritorio
/temp
metadata_path=C:/Documents and Settings/si/Escritorio/temp
```

Los datos más significativos de la configuración es que se ha usado una política hash y un acceso a fichero directo desde el sistema de ficheros.

El binario del servidor de E/S se encuentra alojado en una carpeta compartida por todos los computadores a través de red (ioserver_exec_path). Las rutas para los servidores para el alojamiento de los datos y metadatos son direcciones accesibles y comunes por todos los nodos de manera local (ioserver_data_path y metadata_path).

7.1.1. RENDIMIENTO ALMACENAMIENTO LOCAL

La motivación de la primera prueba realizada es conocer el actual rendimiento que ofrecen los sistemas de ficheros NTFS en Windows® y Ext3 en Linux.

Mediante la realización de este estudio se quiere comprobar las diferencias existentes entre los sistemas de ficheros, siendo interesante contemplar si existen grandes diferencias entre ellos o si por el contrario, ofrecen un rendimiento similar. Otras de las motivaciones de este estudio ha sido averiguar cuál es el tamaño óptimo de acceso al sistema de ficheros local. Una vez conocido este valor, es posible garantizar que los accesos locales de los servidores de E/S serán los más apropiados.

En la Figura 17 y la Figura 18 se muestra una comparativa de los sistemas de ficheros locales ofrecidos por los sistemas operativos Windows® y Linux.

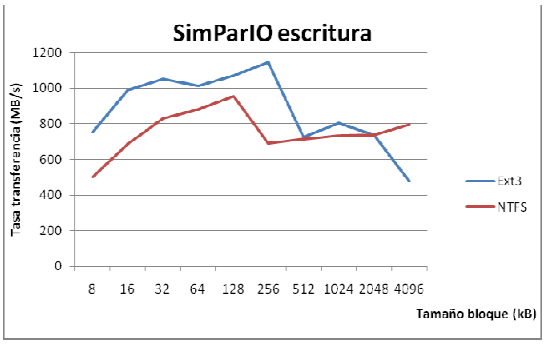


Figura 17: Rendimiento escritura discos locales

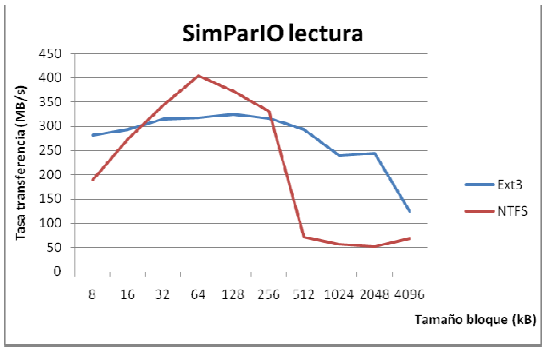


Figura 18: Rendimiento lectura discos locales

Tras los resultados ofrecidos, se puede comprobar que el rendimiento del sistema de fichero Ext3 (Linux) es mayor que el ofrecido por el sistema de ficheros de Windows® (NTFS). Para ambos casos se observa un gran rendimiento en accesos locales. Esto es debido a las optimizaciones que incorporan estos sistemas de ficheros, como caches (*caching*) y lecturas anticipada (*prefetching*). En evaluaciones futuras presentadas en este capítulo, podremos observar como estas optimizaciones solo son aplicadas cuando se accede al fichero por un solo proceso.

Otro detalle que se puede observar es como el sistema Ext3 tiene un rendimiento mucho más estable tanto para las operaciones de lectura como para escritura, mientras que el sistema de ficheros NTFS, al sobrepasar el tamaño de bloque de 256 KBytes, reduce su rendimiento ofreciendo unos resultados bastante malos.

7.1.2. EVALUACIÓN DE ESCALABILIDAD DE AHPIOS

Durante este apartado, se realiza un estudio de la escalabilidad ofrecida por el sistema de E/S paralela AHPIOS, comparando el rendimiento ofrecido utilizando un número diferente de servidores.

La motivación de la realización de la siguiente prueba y análisis viene dado por:

- Medir el rendimiento según distintas configuraciones de los nodos.
- Comprender de mejor manera los resultados ofrecidos por el conjunto de pruebas realizadas con posterioridad en los siguientes apartados.

En la Figura 19 y la Figura 20, se muestra el rendimiento ofrecido en el laboratorio bajo la plataforma Windows®. Para realizar este estudio se ha usado la herramienta SimParIO, con distintas configuraciones de números de servidores de E/S. Como muestran las figuras, se ha realizado una evaluación en dos escenarios distintos: en el primero de ellos se ha comprobado el rendimiento para 16 procesos y para el segundo, con 20 procesos, de tal forma, que todos los nodos de cómputo tengan un servidor en la misma máquina.

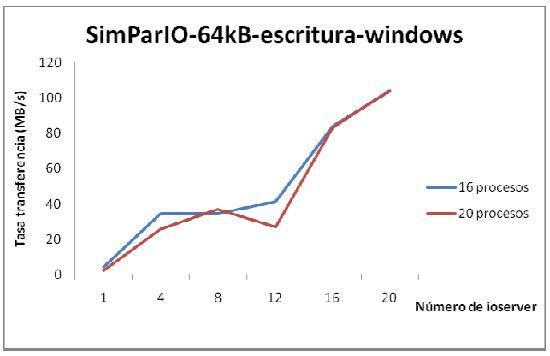


Figura 19: Escalabilidad en escritura de AHPIOS en Windows

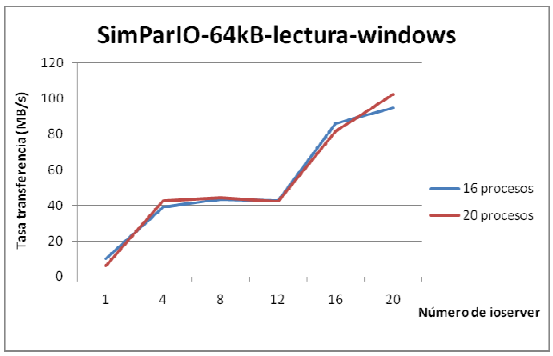


Figura 20: Escalabilidad en lectura de AHPIOS en Windows

Como se puede observar de los resultados ofrecidos, de la ejecución del sistema de evaluación con las distintas configuraciones del laboratorio, se obtiene un rendimiento mucho mayor utilizando un mayor número de servidores de E/S. Esto es debido a que al utilizar un mayor número de servidores, se distribuye la carga de transmisión de datos entre varios nodos de cómputo. De este modo se evita el "cuello de botella" producido cuando la ejecución es realizada con un único servidor de E/S. En el caso de las lecturas, solo se alcanza el grado de escalabilidad cuando se usan más de 12 servidores de E/S. Como se presenta en este mismo estudio, en las plataformas Linux AHPIOS no tiene el mismo comportamiento. Se propone como trabajo futuro estudiar al comportamiento para el intervalo de 4 a 12 procesos. Inicialmente se puede decir que el mecanismo de ejecución de los procesos dinámicos puede afectar el rendimiento, comparando con el sistema operativo Linux.

A continuación en la Figura 21 y la Figura 22, se ofrece el rendimiento obtenido al realizar la misma prueba pero bajo el sistema operativo Linux.

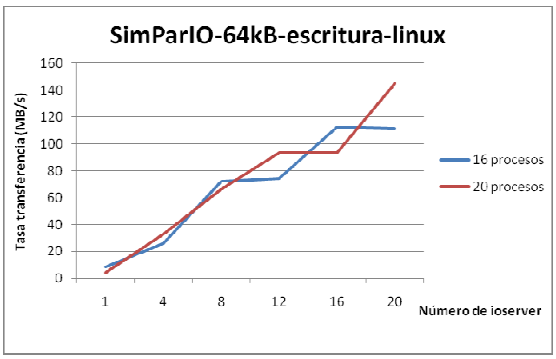


Figura 21: Escalabilidad en escritura de AHPIOS en Linux

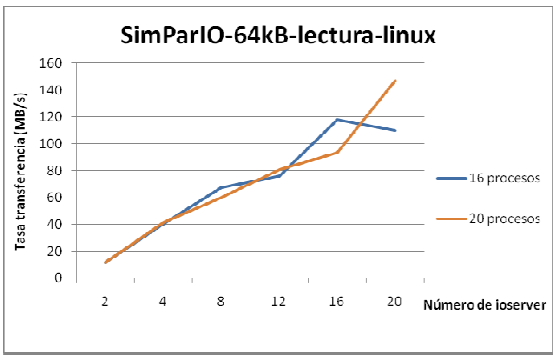


Figura 22: Escalabilidad en lectura de AHPIOS en Linux

Al igual que pasaba con la prueba anterior, se observa como el rendimiento mejora a medida que se aumenta el número de servidores de E/S, motivado por la distribución de la carga de transmisión de datos.

Tras la realización de este conjunto de pruebas, se puede concluir que el sistema de E/S paralela AHPIOS, es fácilmente escalable, proporcionando un mayor rendimiento. A pesar de ello, en Linux se observa una mayor tasa de transferencia. Trabajos futuros podrían estar orientados en estudiar porqué en las plataformas Windows se obtiene una menor tasa de transferencia de datos.

7.1.3. EVALUACIÓN DE CIFS Y AHPIOS

En este apartado se realiza un estudio sobre la diferencia de rendimiento al ejecutar las herramientas de evaluación descritas bajo la plataforma Windows®. Se evaluarán dos sistemas de E/S distintos, por un lado el ofrecido por el sistema operativo Windows® (SAMBA o CIFS) y por otro sistema de E/S paralela migrado (AHPIOS).

A continuación se muestra en la Figura 23 y la Figura 24 los resultados obtenidos al aumentar el número de procesos que intervienen en la ejecución de la aplicación en ambos sistemas. En esta evaluación se ha fijado el número de servidores de E/S en 4.

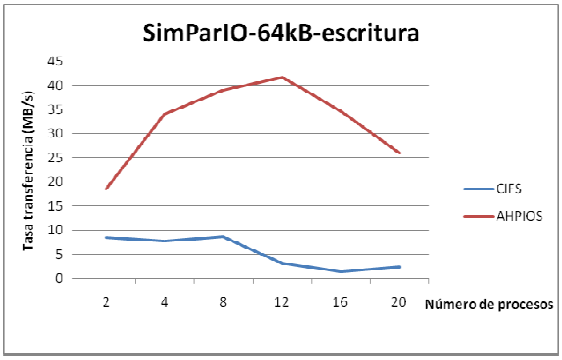


Figura 23: Comparativa escritura CIFS y AHPIOS SimParIO 64 kB

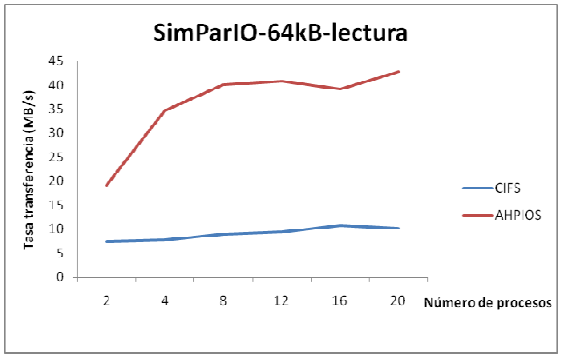


Figura 24: Comparativa lectura CIFS y AHPIOS SimParIO 64 kB

En la Figura 25 y Figura 26, se muestran los resultados obtenidos en ambos sistemas a medida que se aumenta el tamaño del bloque.

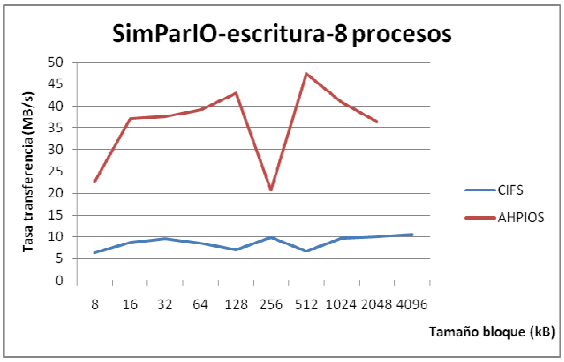


Figura 25: Comparativa escritura CIFS y AHPIOS SimParIO 8 procesos

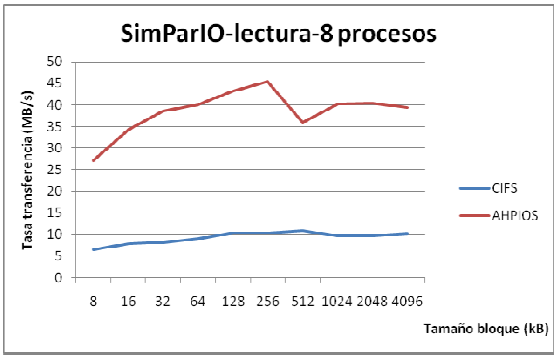


Figura 26: Comparativa lectura CIFS y AHPIOS SimParIO 8 proceso

Tras los resultados arrojados se puede concluir que AHPIOS ofrece resultados mejores en comparación con el sistema de ficheros distribuido existente en la actualidad en la plataforma Windows®.

Lo primero que observa es como el sistema de AHPIOS sufre la misma tendencia que el sistema de ficheros distribuido CIFS, a medida que se aumenta el número de procesos que intervienen en la ejecución de la aplicación. Otra observación es el incremento de rendimiento de la E/S, consiguiéndose una media de un 300% de mejora con respecto al sistema de ficheros CIFS.

Por último se puede observar como el sistema de E/S paralela ofrece un rendimiento muy estable para la lectura, en cambio la escritura ofrece siempre una caída de rendimiento entorno cuando el tamaño de acceso es 256 KBytes (dependiendo del número de procesos que intervengan).

7.1.4. EVALUACIÓN DE AHPIOS EN DISTINTAS PLATAFORMAS

En esta sección se ha evaluado como el sistema de E/S paralela AHPIOS se comporta sobre distintas plataformas, Windows® y Linux. El objetivo de la realización de esta serie de evaluaciones es comprobar si el sistema de E/S paralela AHPIOS, obtiene mejores resultados según el sistema operativo sobre el que esté trabajando, o en contra, se obtiene un rendimiento parecido. Esta evaluación arrogará si el sistema de E/S AHPIOS es una alternativa viable a los sistemas de ficheros actuales (GPFS y Lustre).

El escenario de prueba es el mismo que se comentó en la sección anterior. Para todas las ejecuciones se ha usado un número fijo de servidores de E/S de 4.

A continuación se muestra en la Figura 27 y la Figura 28, los resultados que se obtienen al aumentar el número de procesos que intervienen en la ejecución de la aplicación en ambos sistemas.

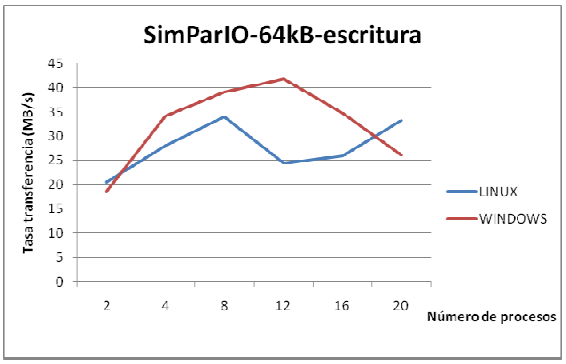


Figura 27: Comparativa escritura Linux y Windows® SimParIO 64 kB

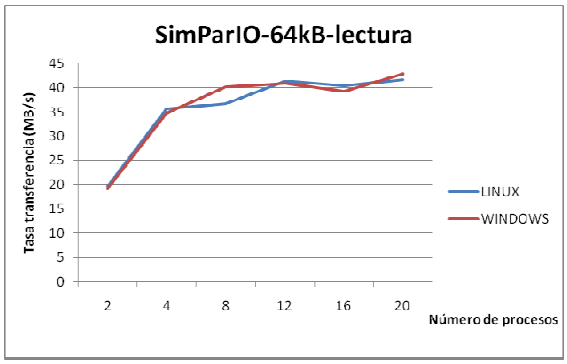


Figura 28: Comparativa lectura Linux y Windows® SimParIO 64 kB

Se puede observar que el caso en el que AHPIOS obtiene mejores resultados tanto para escrituras como para lecturas, es cuando se ejecuta sobre 8 procesos. Para reafirmar este suceso, en las ficheras siguientes se muestra una ejecución detallada, donde se ha variado el tamaño de acceso.

En la Figura 29 y Figura 30, se muestran los resultados obtenidos en ambos sistemas a medida que se aumenta el tamaño del bloque.

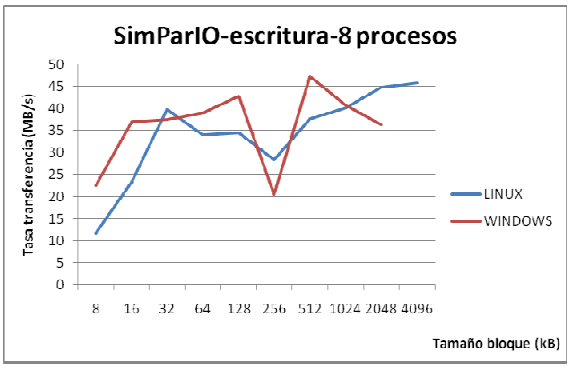


Figura 29: Comparativa escritura Linux y Windows® SimParIO 8 procesos y 4 servidores de E/S

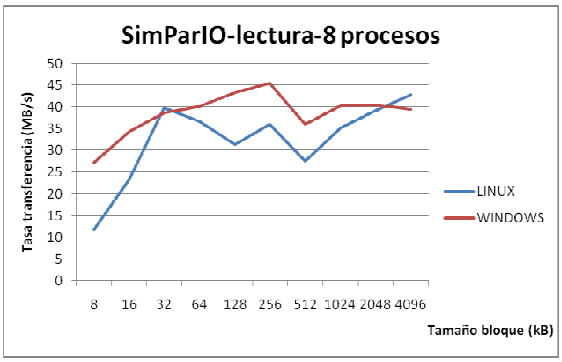


Figura 30: Comparativa lectura Linux y Windows® SimParIO 8 procesos y 4 servidores de E/S

Tras los resultados arrojados se puede observar en primer lugar como el sistema de E/S paralela AHPIOS ofrece un gran rendimiento en ambos sistemas. Otro detalle que se observa es como bajo el sistema operativo Windows®, AHPIOS ofrece un rendimiento mayor, aunque muy semejante al que ofrece bajo el sistema operativo Linux. Esta característica hace de AHPIOS una solución muy adecuada para ambos entornos.

Se puede observar como AHPIOS ofrece un buen rendimiento para las lecturas en ambos sistemas operativos. Por otro lado, en las escrituras ofrece un rendimiento bastante estable cuando se ejecuta bajo el sistema operativo Linux, mientras que su comportamiento es más irregular cuando trabaja en Windows®.

A primera vista es complejo argumentar los buenos resultados de AHPIOS. Inicialmente estos resultados pueden deberse a la gestión de los recursos por parte del sistemas operativo. Además se ha observado que la instalación del sistema Linux de las aulas tiene ejecutándose un mayor número de servicios (NFS, monitores del sistema, etc). En conclusión se complicado sacar conclusiones sobre ambos sistemas.

7.2 CONCLUSIONES

Tras la realización del análisis ofrecido durante los puntos anteriores, a continuación se realiza unas conclusiones sobre todo el sistema de E/S paralela.

- Lo primero que se ha advertido, es como al ejecutar las benchmarks con un sistema de ficheros distribuidos como CIFS o NFS, se produce el efecto de cuello de botella o crisis de la E/S, solucionándose este inconveniente mediante la ejecución con AHPIOS.
- Otro factor que se ha advertido es la capacidad de escalabilidad que ofrece AHPIOS, tanto en el sistema Linux como Windows®, aumentando el rendimiento a medida que se incrementaba el número de servidores existentes.
- Tras la comparación entre el sistema de ficheros distribuidos CIFS y AHPIOS se advierte de un incremento de rendimiento en la E/S de un 200% en el peor de los caso y de un 400% en el mejor caso, obteniéndose una media de incremento de rendimiento entorno al 300%. Se ha comprobado que el uso de APHIOS es altamente recomendable en comparación con el sistema CIFS.
- Tras la evaluación y comparación del sistema AHPIOS bajo ambos sistemas operativos (Linux y Windows®), se ha demostrado que el rendimiento es ligeramente superior bajo el sistema Windows, siendo inicialmente sorprendente dicho resultado debido a que en la comparación de los sistemas de ficheros locales tiene un ligero mayor rendimiento el sistema de ficheros de Linux frente a Windows®. A pesar de esta pequeña mejoría bajo el sistema Windows, no se considera lo suficiente significativa como para afirmar el mejor rendimiento.

- Se ha observado un comportamiento distinto cuando se crean los procesos dinámicos (en este caso los servidores de E/S). Mientras que en Linux el primer proceso dinámico es ejecutado en la primera máquina disponible, en Windows se ejecuta el primero de ellos en la máquina desde que se inicia la ejecución. Esta características puede afectar la escalabilidad como se ha observado en la evaluación.
- Por último, comentar la localización de un problema en el código nativo de la distribución de MPICH. Este problema afecta a la gestión de los tipos de datos, la cual impide la repartición de los datos no contiguos entre los procesos bajo la plataforma Windows. Este problema se ha hecho saber al grupo de desarrollo del sistema y se ha abierto una incidencia dentro de su sistema para su posible solución. Se han ejecutado un mayor tipo de pruebas en entorno Linux. Próximamente se espera poder realizar las comparaciones restantes.

8. CONCLUSIONES Y TRABAJOS FUTUROS

En este capítulo se revisan los objetivos planteados que fueron marcados al inicio del proyecto, asi como del grado de cumplimiento de los mismos. Por último se proponen futuras mejoras y líneas de trabajo.

Además se comentará la metodología seguida durante la realización del proyecto, así como el presupuesto final para la realización del mismo.

8.1. CONCLUSIONES

Como objetivo principal del proyecto se ha conseguido migrar el sistema de E/S paralela AHPIOS hacia el sistema operativo Windows®, permitiendo el ahorro de tiempo en la ejecución de aplicaciones paralelas.

El proyecto no ha quedado finalizado hasta que se ha comprobado que todos los objetos marcados han sido completados:

- Se ha realizado un estudio de los sistemas de ficheros paralelos que existen en la actualidad para el sistema operativo Windows®.
- Se ha indagado sobre las posibilidades de Windows® como sistema HPC (High Performance Computing).
- Se ha conseguido migrar una solución de un sistema de E/S paralelo desarrollado en Linux a un entorno Windows®.
- Se ha evaluado el sistema y comparado bajo distintos sistemas operativos para comprobar cuál es más recomendable.

Además de los objetivos afines a la realización de este proyecto, se derivan objetivos meramente académicos. Gracias a la realización de este proyecto se han profundizado en los siguientes aspectos:

- Se ha aprendido a gestionar un proyecto grande que ya ha sido iniciado, además de la dificultades que entraña una migración y las diferencias existentes entre proyectos en Linux y en Windows®.
- Se ha profundizado en las características del lenguaje C y del entorno de programación Visual Studio .Net.
- Estudio sobre funcionamiento, características, limitaciones, etc. de los sistemas de ficheros paralelos.
- Profundizado en las posibilidades que ofrece MPI.
- Aprendido una metodología de evaluación de sistemas (benchmarking).

8.2. TRABAJOS FUTUROS

Para ampliar las posibilidades que puede ofrecer este Proyecto Fin de Carrera, se proponen las siguientes tareas:

- Integrar servicios web dentro de la aplicación para permitir trabajar en entornos como Grid Computing o Cloud Computing, Este trabajo podría permitir realizar servicios de cómputo distribuido entre recursos de hardware heterogéneos comunicados entre sí mediantes redes de área extensa como Internet.

- Implementar un sistema completo de metadatos completamente funcional, incorporado al actual sistema de E/S, con el objetivo de gestionar los datos que se encuentran en el sistema de ficheros virtual, de manera transparente al usuario. Para esta tarea se podría usar herramientas como FUSE.

- Migrar el tipo de acceso al fichero mediante el mapeo del fichero en memoria (mmap en POSIX). El mapeo de memoria en la plataforma .Net no respeta los estándares de POSIX, por lo que es necesario un estudio de las posibles alternativas, tanto dentro del API de .NET como librerías externas.

- Realizar un mayor número de pruebas y evaluaciones. Entre ellas, ejecutar aplicaciones implementadas en distintos lenguajes de programación, como puede ser fortran. Fortran es un lenguaje de programación muy extendido dentro de la comunidad científica.

8.3. MÉTODO DE TRABAJO Y PRESUPUESTO

En esta sección se describe la metodología utilizada en el proyecto, así como la organización y planificación de las tareas que se van a llevar a cabo; además, se especifican los recursos humanos y materiales necesarios para la realización del proyecto. Para finalizar, se ofrece un presupuesto que refleja el coste total y de mercado que supondría el desarrollo del proyecto.

8.3.1. METODOLOGÍA DE TRABAJO

Para el desarrollo del proyecto se ha utilizado el ciclo de vida incremental o de versiones sucesivas. La elección de este ciclo de vida se debe además de porque el proyecto es de gran tamaño, porque algunas de las funcionalidades del proyecto deben estar operativas antes que otras y además, este método permite validar el sistema a medida que se construye.

Con este método, cada versión o fase es un sistema funcional capaz de realizar progresivamente la función deseada.

Como se puede observar en la Figura 31, el ciclo de vida incremental posee una fase de análisis inicial y varias fases de diseño, implementación y evaluación de las pruebas de forma incremental.

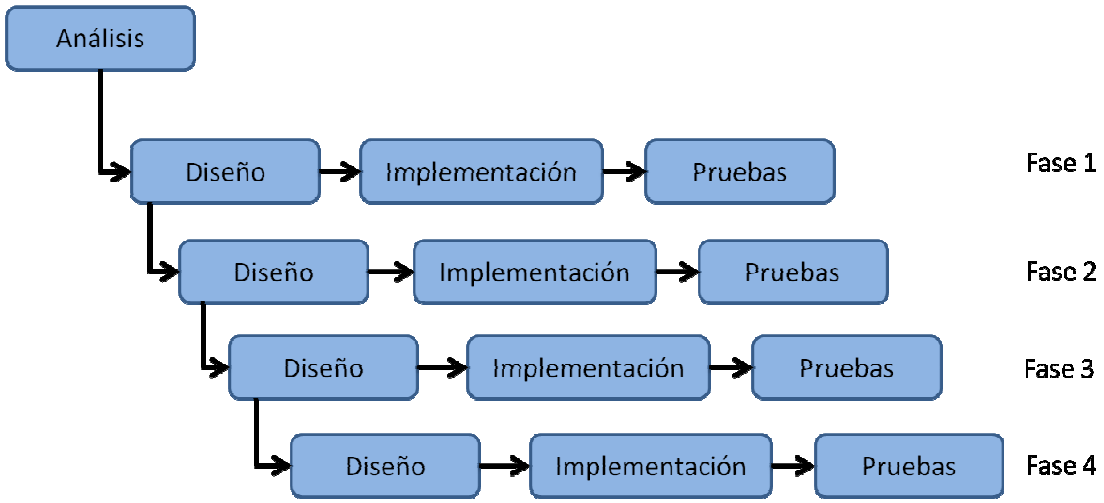


Figura 31: Ciclo de vida incremental

A continuación se resumen los objetivos de cada una de las fases del diseño incremental.

- Análisis:

  Se realiza un estudio de las posibilidades que ofrece el sistema desarrollado dentro del ámbito actual, y se comprueba la disponibilidad tecnológica que existe en la actualidad.

- Fase 1:

  Una vez realizado un estudio previo se lleva a cabo el proceso de migración del sistema desarrollado en Linux a la nueva plataforma Windows®.

- Fase 2:

  Una vez migrado el sistema, se procede a realizar la configuración e instalación de los procesos necesarios, para el correcto funcionamiento en el laboratorio.

- Fase 3:

  Se procede a la ejecución de las pruebas que se han considerado de relevancia, almacenando los datos obtenidos en una organización de carpetas representativas.

- Fase 4:

Se realiza una evaluación final de los resultados de las pruebas hechas en la fase anterior, pudiendo obtener unas conclusiones satisfactorias de esta evaluación.

8.3.2. PRESUPUESTO

En este apartado se realiza el desglose del coste asociado que tiene la realización del proyecto.

- **Gastos de personal imputables al proyecto**

Para realizar el cálculo del presupuesto se han tenido en cuenta las siguientes consideraciones:

- Jornada laboral de 5 horas.
- 5 días laborables a la semana (excluidos fines de semana y festivos).
- Las vacaciones y días festivos considerados en la planificación son:
  - 20 y 21 de Marzo: jueves y viernes Santo.
  - 1 de Mayo: Día del trabajador.
- Periodo de realización:
  - Inicio del proyecto: 08 de Enero de 2009.
  - Finalización del proyecto: 24 de Julio de 2009

- El proyecto requiere un analista y un programador.
    - Coste de personal por hora de trabajo:
    - Analista: 50 €/hora.
    - Analista-programador: 30 €/hora.

Por lo tanto, el cómputo total de horas dedicadas al proyecto es 785 horas, por lo que el proyecto ha tenido una duración total de 157 días reales.

Actividades	Duración (horas)	Recursos	Total (Euros)
Fase 1 (Estudio previo)	75	Analista	3.750 €
Fase 2 (Migración)	240	Analista-Programador	7.200 €
Fase 3 (Instalación de software)	10	Analista-Programador	300 €
Fase 4 (Realización de pruebas)	170	Analista-Programador	5.100 €
Fase 5 (Evaluación de pruebas)	200	Analista	10.000 €
Documentación	90	Analista	4.500 €
Total	785 horas		30.850 €

Tabla 3: Especificación de actividades y coste.

- **Coste del hardware para el desarrollo del sistema**

El despliegue del sistema ha sido realizado con el siguiente hardware:

- Equipo servidor:
  - Pentium III, Dual de 1 Ghz de velocidad y 2 GBytes de memoria.
- 20 nodos:
  - Windows XP
  - AMD Athlon(tm) 64 X2 Dual Core Processor 4200+
  - Cache nivel L1 512 Kbytes
  - Memoria RAM 2 GB
- Un switch de 24 bocas de 100 megabits por segundo.

- **Coste de licencias para el desarrollo del sistema**

CONCEPTO	Importe (€)
Visual Studio.Net 2005	502 €
Total final	**502 €**

Tabla 4: Coste total licencias software

- **Coste total del proyecto**

La siguiente tabla detalla el coste total del proyecto:

Concepto	Importe (€)
Recursos humanos	30.850 €
Licencias	502 €
Total antes de riesgo	**31.352 €**
Riesgo (10%)	3.135,20 €
Total antes de beneficio	**34.387,20 €**
Beneficio (15%)	5.158,08 €
Total final	**39.545,28 €**

Tabla 5: Coste total del proyecto.

El presupuesto final del proyecto se estima en treinta y nueve mil quinientos cuarenta y cinco, con veintiocho céntimos **(39.545,28 €).**

REFERENCIAS

A continuación se exponen los recursos y la documentación que se han consultado y estudiado para el análisis y desarrollo del proyecto. Estas referencias aparecen a lo largo de todo el documento con un número entre corchetes.

[1] Ley de Amdahl: http://home.wlu.edu/~whaleyt/classes/parallel/topics/amdahl.html	02 febrero 2009
[2] Proyecto ARPANET: http://www.iso.port.ac.uk/~mike/docs/internet/internet/node4.html	02 febrero 2009
[3] Protocolo TCP/IP: http://www.yale.edu/pclt/COMM/TCPIP.HTM	02 febrero 2009

[4] BSD (Berkeley) Unix: http://www.freebsd.org/doc/es_ES.ISO8859-1/articles/explaining-bsd/article.html	05 febrero 2009
[5] ARCNET: http://www.arcnet.com/abtarc.htm	05 febrero 2009
[6] Kronenberg, N. P., Levy, H. M., and Strecker, W. D. 1985. VAXclusters (extended abstract): a closely-coupled distributed system. SIGOPS Oper. Syst. Rev. 19, 5 (Dec. 1985), 1. DOI= http://doi.acm.org/10.1145/323627.323628	09 febrero 2009
[7] Tandem Himalaya: http://www.redbarnhpc.com/v2/index.php/cluster-computing/the-history-of-cluster-computing	12 febrero 2009
[8] IBM S/390 Parallel Sysplex: http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=508100	16 febrero 2009
[9] Parallel Virtual Machine (PVM): http://www.csm.ornl.gov/pvm/	16 febrero 2009
[10] Arquitectura Beowulf: http://www.beowulf.org/overview/index.html	17 febrero 2009
[11] RAID: http://www.technick.net/public/code/cp_dpage.php?aiocp_dp=guide_raid	04 marzo 2009
[12] GPFS: http://publib.boulder.ibm.com/epubs/pdf/a7604132.pdf	18 marzo 2009
[13] LUSTRE: http://www.sun.com/software/products/lustre/	18 marzo 2009
[14] PVFS: http://www.pvfs.org/	18 marzo 2009
[15] *Message Passing Interface Forum: MPI: A Message-Passing Interface Standard. (1995)*	
[16] MPI bajo Windows®: http://www.ee.ryerson.ca/~courses/ee8218/MPIWIN.pdf	08 abril 2009
[17] MPI-IO: http://www.nersc.gov/nusers/resources/software/libs/io/mpiio.php	08 abril 2009

[18] ROMIO: http://www.mcs.anl.gov/research/projects/romio/	08 abril 2009
[19] An abstract-Device Interface for Implementing Portable Parallel-I/O Interfaces (6th Symposium on the Frontiers of Massively Parallel Computation, October 1996, pp 180-187, 1996): http://www.mcs.anl.gov/~thakur/papers/adio.pdf	08 abril 2009
[20] MPICH: http://www.mcs.anl.gov/research/projects/mpich2/	08 abril 2009
[21] Performance Evaluation of Remote Memory Access (RMA) Programming on Share Memory Parallel Computers http://www.mcs.anl.gov/~thakur/papers/adio.pdf	14 abril 2009
[22] A performance Study of Two-Phase-I/O: http://www.umcs.maine.edu/~dickens/pubs/292.21.ps	14 abril 2009
[23] MPE: http://hpc.usg.utk.edu/docs/mpich-intel/mpeman.pdf	15 abril 2009
[24] JUMPSHOT: http://www.mcs.anl.gov/research/projects/mpi/www/www1/Jumpshots	15 abril 2009
[25] Windows HPC server 2008: http://www.microsoft.com/hpc/en/us/default.aspx	05 mayo 2009
[26] NTFS: http://www.ntfs.com/	05 mayo 2009
[27] SAMBA: http://www.samba.org/	05 mayo 2009
[28] Professional .NET framework 2.0 / Joe Duffy (Wiley), 2006	
[29] Bechmark: http://es.tldp.org/COMO-INSFLUG/es/pdf/Benchmarking-COMO.pdf	12 mayo 2009
[30] An MPI implementation of ad-hoc scalable parallel I/O (Proceedings of the 2008 14th IEEE International Conference on Parallel and Distributed Systems, pp: 253 – 260)	

APÉNDICE I:
MANUAL DE INSTALACIÓN DE MPI

A continuación se detallan los requisitos y pasos necesarios para realizar una instalación satisfactoria de MPICH2 en un sistema operativo Windows.

- **Requisitos**

1) MS Development Environment.
2) Visual Studio o gcc para compilar los programas MPI en C/C++.
3) Intel Fortran 8.0 para compilar los programas MPI en Fortran (opcional).
4) Permisos de administrador, para que se pueda instalar el proceso smdp.exe en las máquinas y se permita la copia de las librerías en el directorio de Windows.

- **Instalación**
  - **Distribución binaria**

1) Descargar e instalar en todas las máquinas la distribución MPICH2 de 32 bits o 64 bits según la arquitectura de su computador. El instalador al finalizar copiará las librerías necesarias dentro del directorio windows\system32, además creará la siguiente estructura de directorios en la máquina:
   - `path_instalación\bin`
   - `path_instalación \include`
   - `path_instalación \lib`

2) Tras finalizar el proceso de instalación, editar la variable "path" de las variables de entorno del sistema, para añadir los ejecutables de la distribución, que se encuentran en el directorio 'bin' de la instalación.

3) Si se tiene instalado un firewall, es necesario añadir las excepciones para los programas mpiexec.exe y smpd.exe.

- **Distribución código fuente**

1) Descarga el fichero con el código y descomprimirlo.
2) Ejecutar la consola del sistema y ejecutar "winconfigure.wsf" (necesario tener instalado una distribución de Perl), situado dentro del directorio donde se ha descomprimido todo el código fuente. Si no se dispone de fortran instalado añadir la opción al ejecutar de --remove-fortran.
3) Tras terminar la ejecución satisfactoriamente, abrir la solución "mpich2.sln" con Microsoft Visual Studio.
4) Compilar la solución mpich2 con la configuración ch3sockRelease.
5) Compilar el proyecto mpich2s con la configuración ch3sockRelease.
6) Compilar la solución mpich2 con la configuración Release.
7) Editar la variable "path" de las variables de entorno del sistema, para añadir los ejecutables de la distribución, que se encuentran en el directorio "bin" de la distribución.
8) Si se tiene instalado un firewall, es necesario añadir las excepciones para los programas mpiexec.exe y smpd.exe.

NOTA: Si se quiere únicamente utilizar los nodos para ejecutar los programas MPI, solo es necesario en cada máquina:

- copiar el directorio "bin", añadir los ejecutables que se encuentran dentro en la variable de entorno "path" y ejecutar desde la consola de comandos "smpd.exe -install".
- copiar las librerías de mpich2 (*.dll) que se encuentran en la carpeta "lib" de la distribución, dentro del directorio windows\system32 de cada nodo.

APÉNDICE II: INFRAESTRUCTURA

En este punto se detalla como configurar las máquinas que forman la infraestructura, para que se comuniquen correctamente.

Por cada máquina es imprescindible:

1) Tener correctamente instalado .NET Framework, necesario para que se puedan ejecutar los programas compilados bajo la tecnología .NET.
2) Tener correctamente instalado MPI.
3) Crear una cuenta de usuario común, es decir, en todas las máquinas debe existir la misma cuenta de usuario, donde el login y la contraseña sean comunes en todas.
4) Cada máquina tiene que tener un nombre distinto al resto.
5) Se ha de configurar la interfaz de red, de tal manera que todas las máquinas pertenezcan al mismo subdominio y tengan distinta IP.

Además de la configuración que ha de tener cada máquina, nuestra infraestructura tiene una especificación concreta, la cual sigue el siguiente esquema:

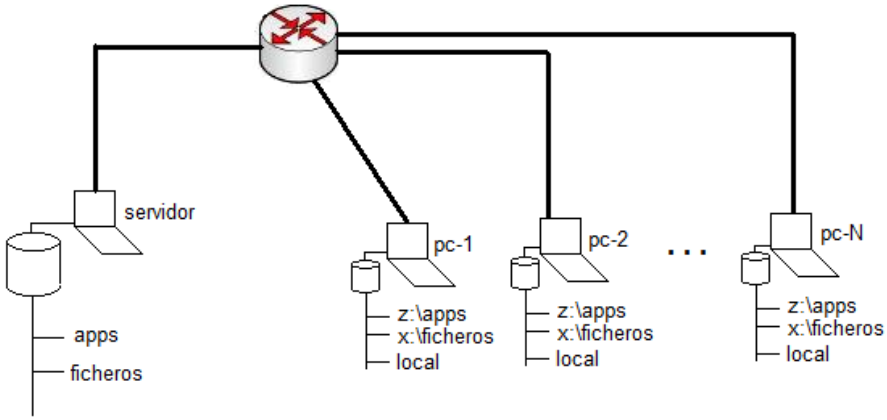


Figura 32: Arquitectura general cluster

Como se puede observar, se ha de configurar una máquina como servidor, la cual albergará las aplicaciones, ficheros, etc. accesibles por el resto de máquinas, dicha configuración se ha de establecer mediante un sistema de compartición de recursos en red, lo cual bajo Windows se configura de la siguiente manera:

- Servidor
  - Compartir carpeta/s en red, en la cual tengan permisos todos los equipos de la red.
- Cliente
  - Conectarse al recurso de red, compartido por el servidor y montarlo como una unidad de disco en red.

APENDICE III:
MANUAL DE USUARIO

En este apartado se indica cómo realizar la compilación y ejecución de los programas paralelos.

- **Compilación programas**

1) Abrir la aplicación Visual Studio.
2) Crear un proyecto de aplicación de consola Win32.
3) Añadir las librerías al proyecto:

   Seleccionar archivos de biblioteca (menú superior derecho) y añadir los ficheros de la carpeta /lib de la instalación de MPI.
4) En la ventana del proyecto, pulsar botón derecho sobre "Archivos de encabezado" y seleccionar añadir elemento existente y añadir todos los ficheros de la carpeta /lib de la instalación de MPI.
5) Crear el programa y compilarlo.

- **Ejecución de programas**

1) Registrar la cuenta de usuario con MPI. Esto es posible realizarlo mediante dos formas diferentes:
   - Utilizando el programa wmpiregister.exe, el cual se encuentra en la carpeta "bin" del directorio donde se instaló la distribución.
   - Ejecutando en la consola del sistema `mpiexec –register`.

   (Nota solo es necesario ejecutar el registro la primera vez que se quiera lanzar una aplicación MPI sobre la máquina y cada vez que se reinicie o encienda)
2) Configurar el fichero maquinas indicando cuales van a participar en la ejecución del programa.
3) Ejecutar el programa MPI a través de la consola del sistema mediante el siguiente comando:
   - `mpiexec -n <nº_maquinas> -machinefile <fichero_maquinas> fichero_aplicacion.`