E/S Paralela

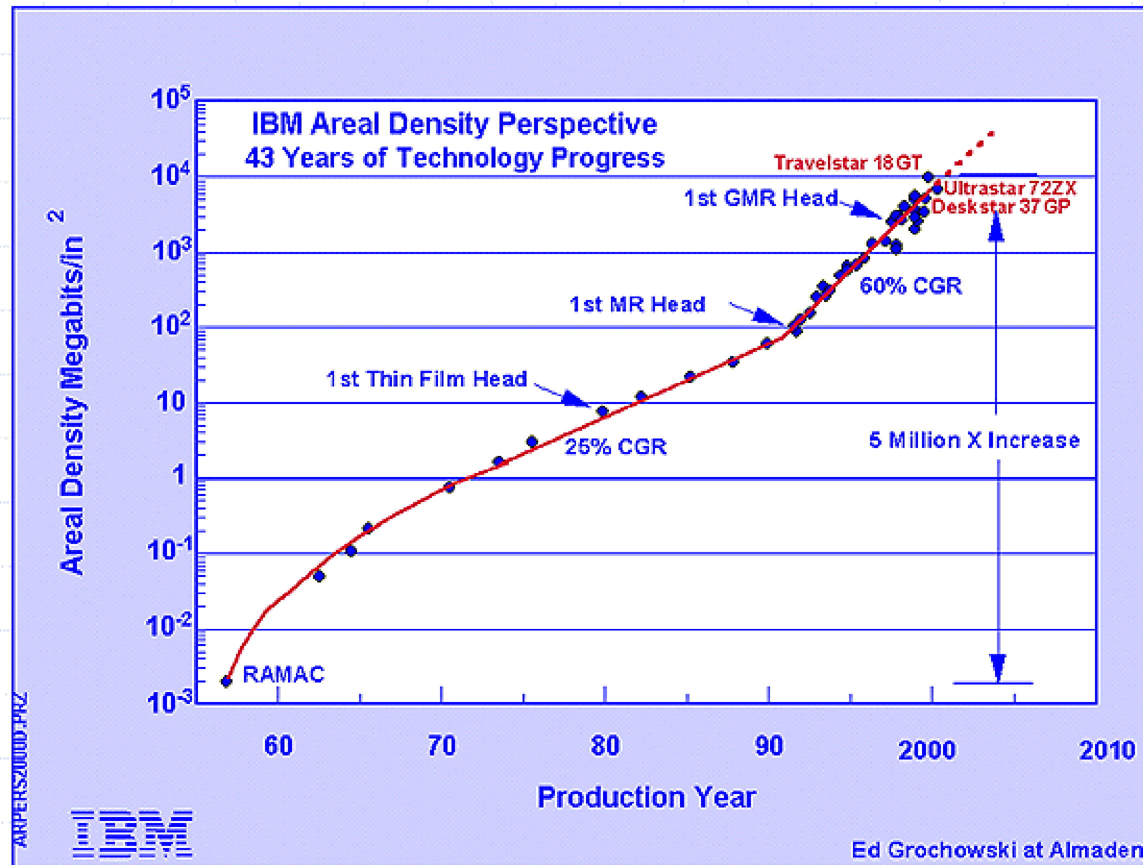
MPI-IO: Un caso de estudio

María S. Pérez (mperez@fi.upm.es)
Dep. Arquit. y Tecnol. de Sist. Informáticos
Facultad de Informática, UPM, Madrid

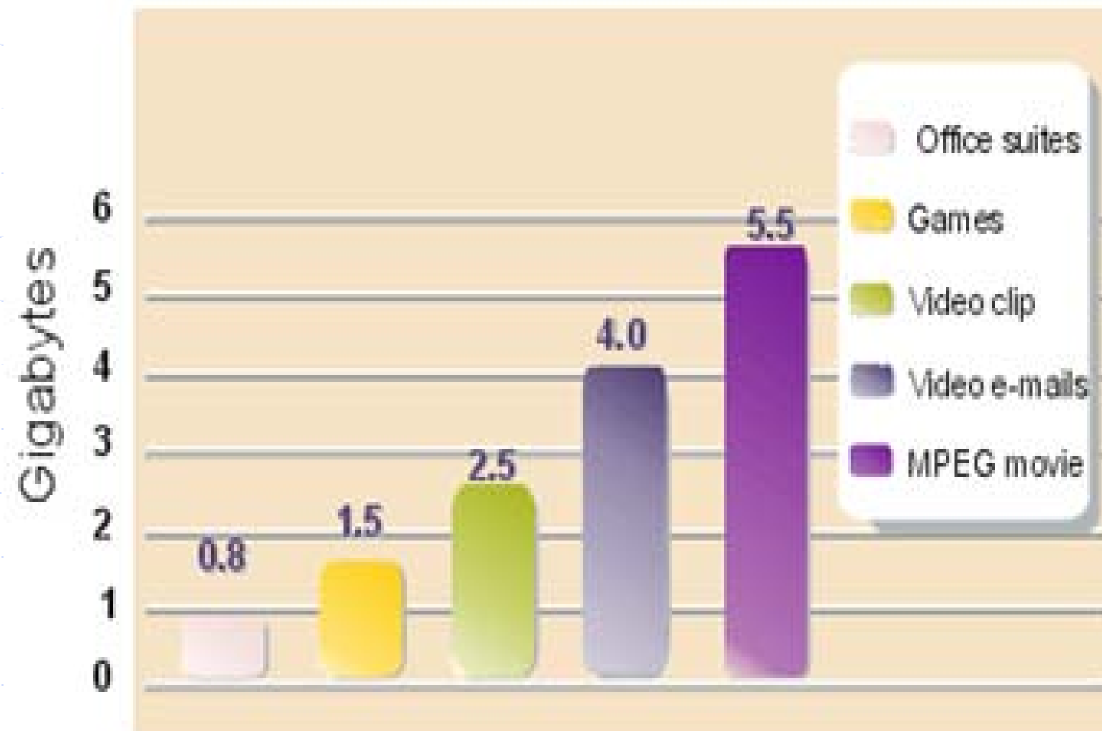
Índice

- Motivación
- Sistemas RAID
- Bibliotecas de E/S y sistemas de ficheros paralelos
- Técnicas de optimización de la E/S
- Interfaz de E/S. MPI-IO
- Conclusiones

Discos duros: Capacidad

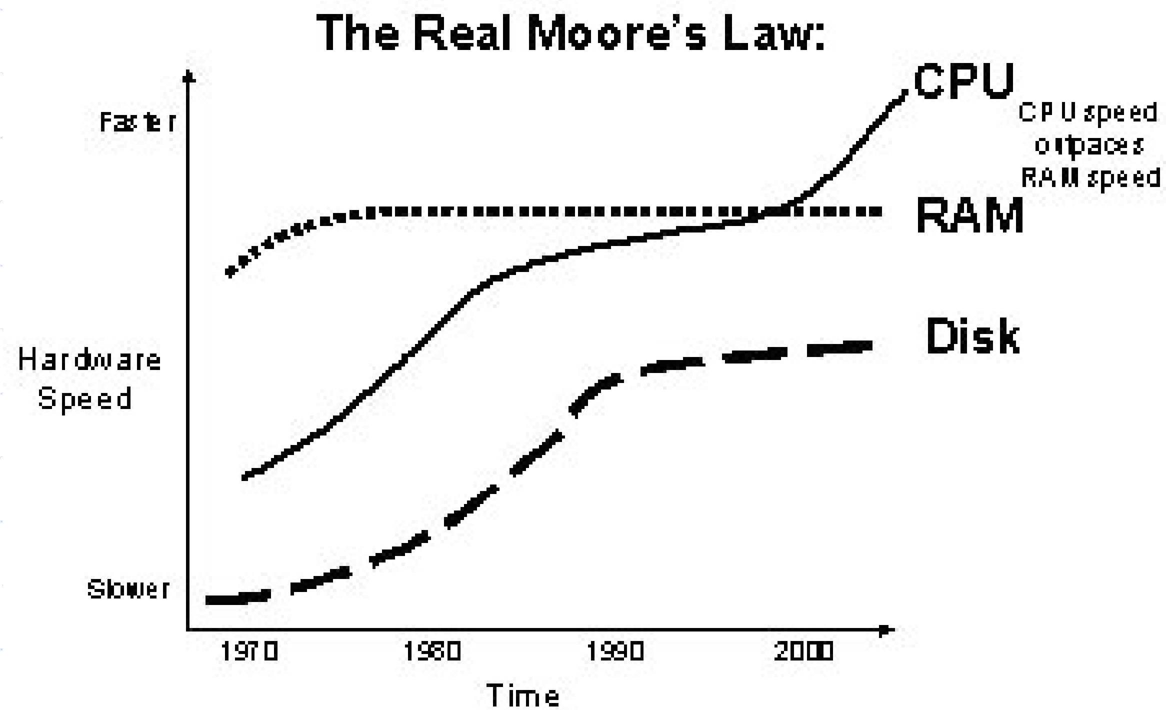


Discos duros: Demanda de las aplicaciones



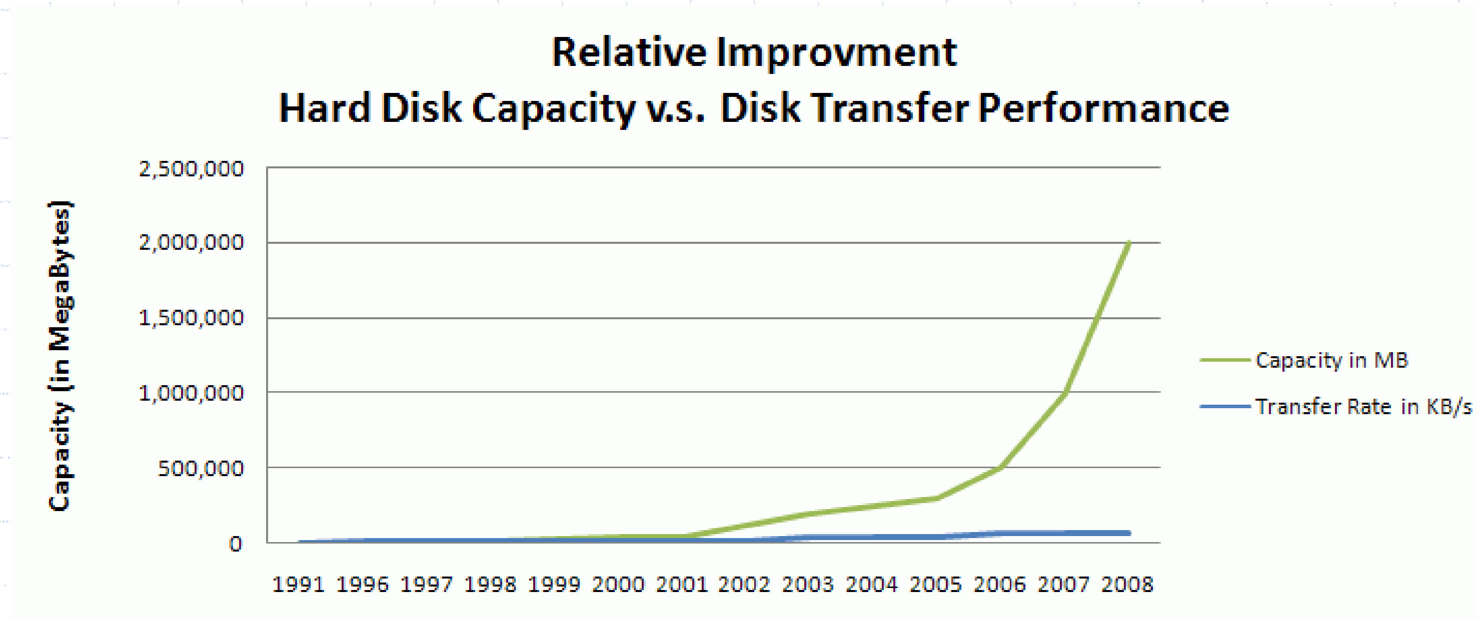
Sources: Maxtor Corporation and *Trend*FOCUS

Discos duros: Rendimiento



Source: Oracle

Discos duros: Capacidad vs Rendimiento



Motivación de la E/S paralela

- Problema:
  - Capacidad: Se incrementa a gran velocidad (las aplicaciones también).
  - Ancho de banda y latencia: Depende de componentes mecánicos y, por tanto, no se incrementa a la velocidad de los componentes electrónicos.
- Ventajas:
  - Los dispositivos son baratos
- Solución: Utilización de paralelismo

Crisis de la E/S

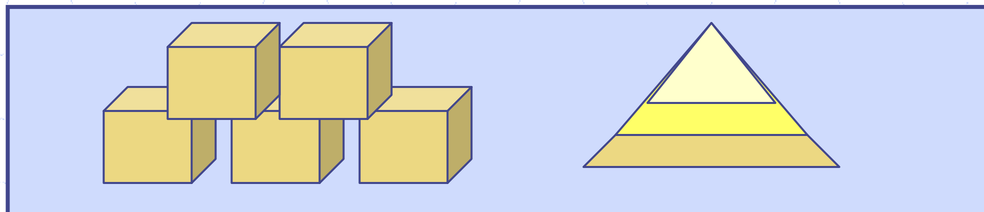
- Crisis originada por el desequilibrio entre el tiempo de cómputo y de E/S.
- Aún más acusado en sistemas multiprocesadores.
- Soluciones:
  - Utilización de paralelismo en el sistema de E/S
  - Redes de almacenamiento (NAS)

Sistemas de ficheros distribuidos vs paralelos

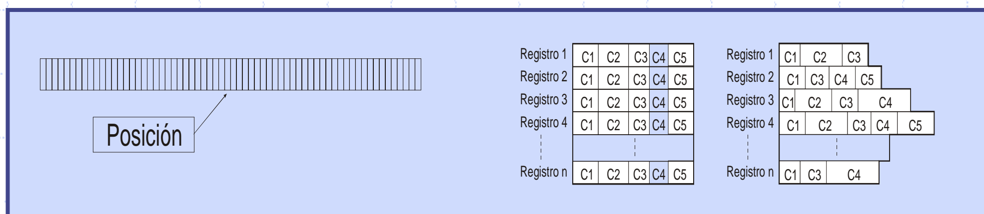
- Muchos de los conceptos y de las ideas en que se basan son iguales.
- Generalmente:
  - E/S paralela: relacionada con el rendimiento
  - E/S distribuida: relacionada con la disponibilidad

Jerarquía de la E/S paralela

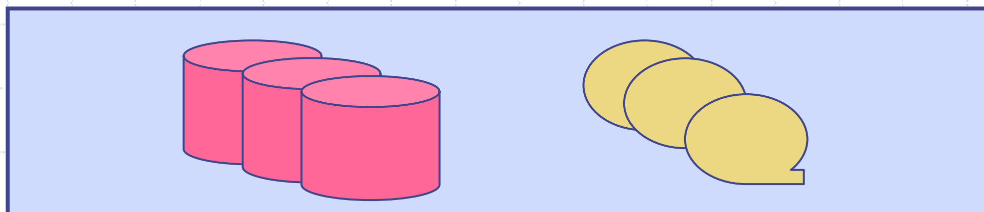
Aplicaciones paralelas



Bibliotecas paralelas



Sistemas de ficheros/ Sistemas operativos paralelos



Dispositivos paralelos

Índice

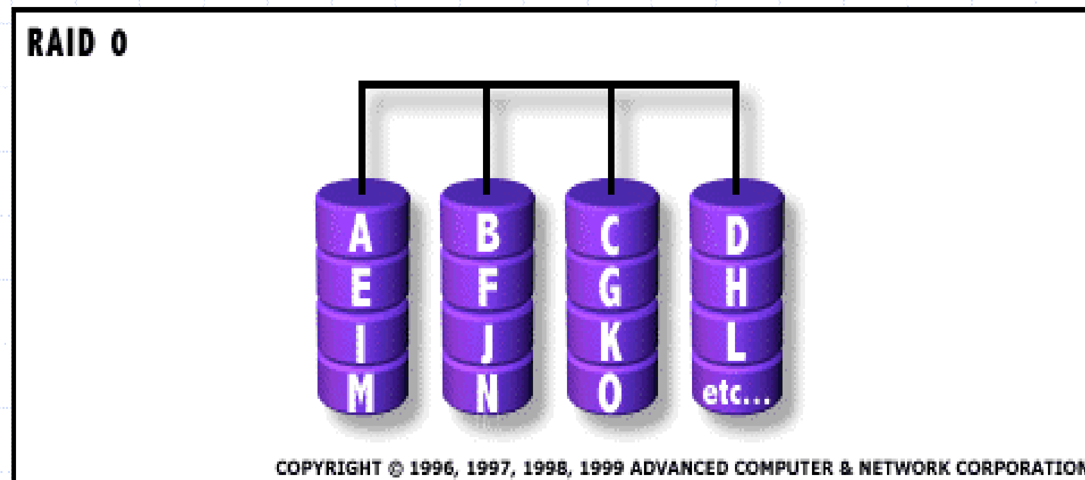
- Motivación
- Sistemas RAID
- Bibliotecas de E/S y sistemas de ficheros paralelos
- Técnicas de optimización de la E/S
- Interfaz de E/S. MPI-IO
- Conclusiones

E/S paralela

- La idea original procede del sistema RAID (Redundant Array of Inexpensive Disks).
- RAID:
  - Objetivo: Utilizar varios discos como un solo dispositivo, a fin de incrementar el ancho de banda del sistema de almacenamiento.
  - Problema: El número de discos utilizados es proporcional a la probabilidad de fallos
    - Solución: Proporcionar tolerancia a fallos

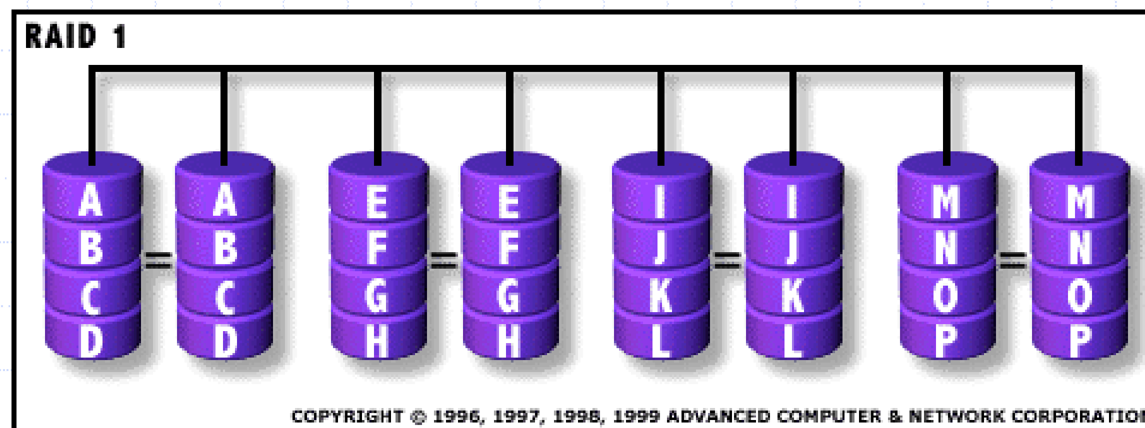
RAID 0

- Ventajas:
  - Alto ancho de banda y capacidad
- Desventajas:
  - No tolerancia a fallos



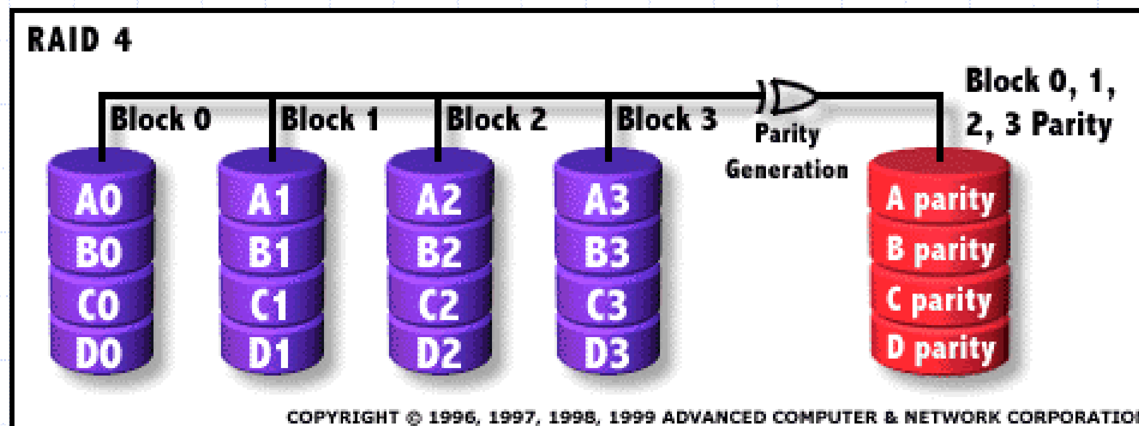
RAID 1

- Ventajas:
  - Tolerancia a fallos
  - Posibles optimizaciones de lectura
- Desventajas:
  - Espacio malgastado
  - Menor paralelismo que RAID 0



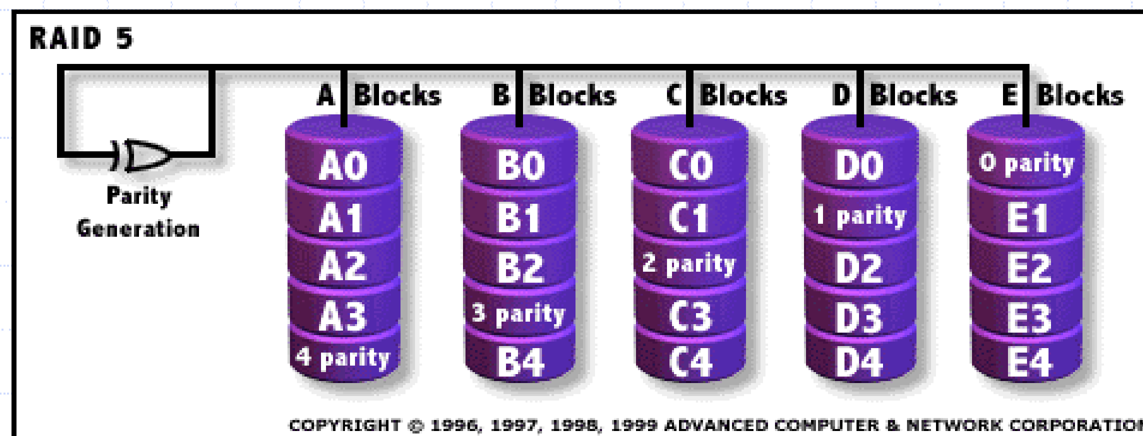
RAID 4

- Ventajas:
  - Tolerancia a fallos (permite fallo de un único disco)
  - Paralelismo en la operación de lectura
- Desventajas:
  - No paralelismo en la operación de escritura (siempre utiliza el disco de paridad)



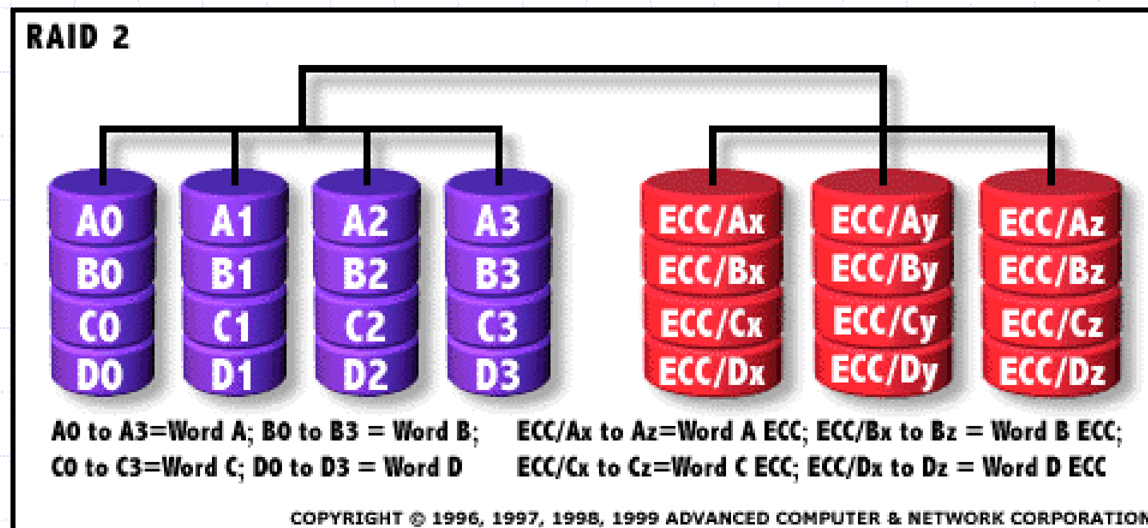
RAID 5

- Ventajas:
  - Alto ancho de banda tanto de lectura como escritura
  - Tolerancia a fallos (sólo para fallo de un disco)
- Desventajas:
  - Problema con las escrituras pequeñas (*small writes*)



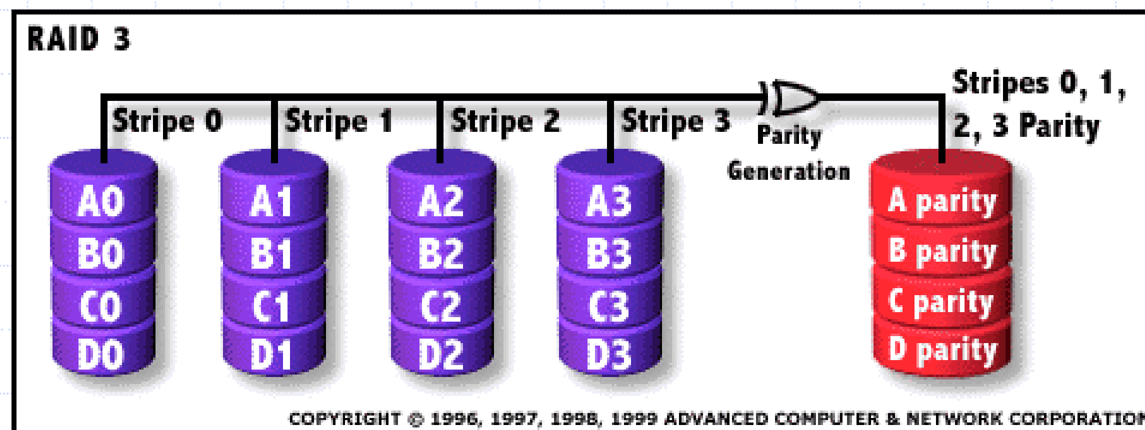
RAID 2

- Ventajas:
  - Permite error
  - Alto rendimiento
- Desventajas:
  - Alto *ratio* de discos ECC respecto a discos de datos
  - No existen implementaciones comerciales



RAID 3

- Ventajas:
  - Alto rendimiento
  - Bajo *ratio* de discos de paridad respecto a discos de datos
- Inconvenientes:
  - El diseño del controlador es complejo
  - Operación de escritura



Otras configuraciones RAID

- Raid 6:
  - Permite el fallo de dos discos mediante el uso de más bloques de paridad
- Raid 7:
  - Optimizado para altas velocidades de transferencia
  - Solución propietaria
- Raid 53:
  - Combinación de RAID 0 y RAID 3
- Raid 10:
  - Un array distribuido cuyos segmentos son arrays RAID 1
- Raid 0+1:
  - Un array espejo cuyos segmentos son arrays RAID 0

Problema de las “escrituras pequeñas” (*small-write problem*)

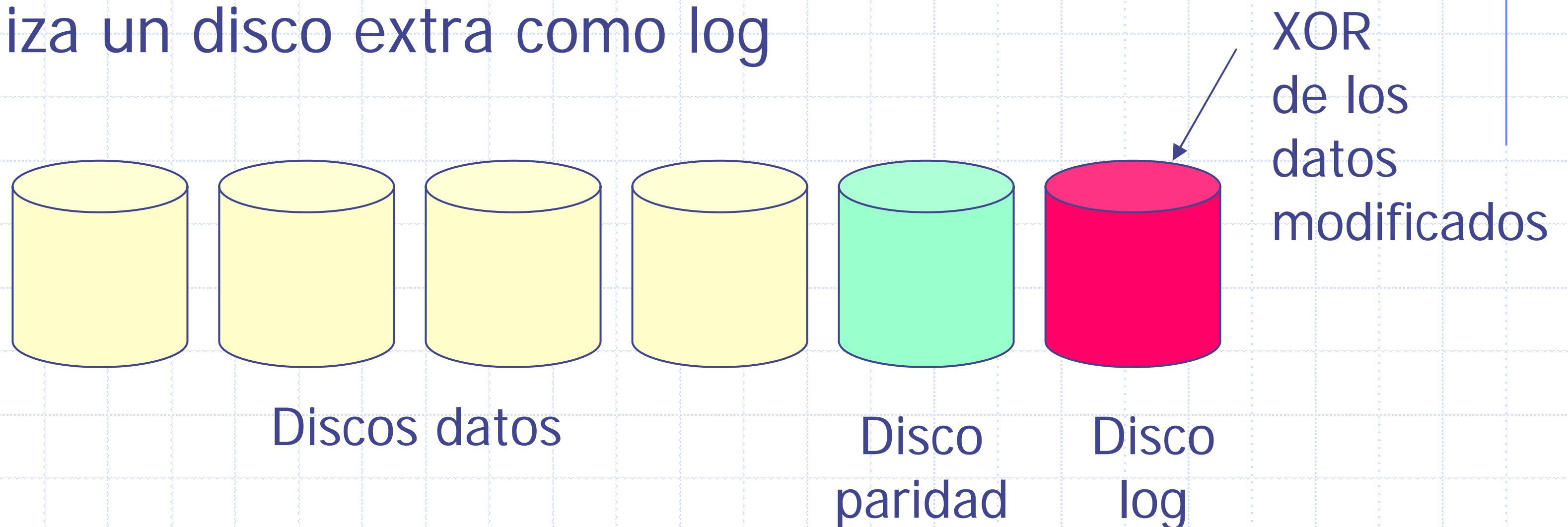
- Pasos:
  - Leer datos antiguos
  - Calcular la nueva paridad
  - Escribir los nuevos datos y la nueva paridad ⇒ *small-write problem*
- Soluciones:
  - Nuevas jerarquías RAID (ej. HP AutoRAID)
  - Uso de logs (ej. xFS, Zebra)
  - *Parity logging*

HP AutoRaid

- Define dos niveles de RAID:
  - RAID 1:
    - Lecturas y escrituras rapidas ☺
    - Tolerante a fallos ☺
    - Requiere mucho espacio de almacenamiento ☹
  - RAID 5:
    - Lecturas rapidas ☺
    - Tolerante a fallos ☺
    - Mínimo espacio de almacenamiento ☺
    - Escrituras pequeñas lentas ☹
- *"The HP AutoRAID hierarchical storage system"*
  John Wilkes, Richard Golding, Carl Staelin, Tim Sullivan
  ACM Transactions on Computer Systems (TOCS)
  Volume 14 , Issue 1, February 1996
  Pages: 108 – 136

Parity Logging

- Utiliza un disco extra como log



- Problema de *small-writes*:
  - Escritura de bloques de datos igual que en RAID 4/5
  - Escritura de la paridad en el disco *log* (escritura secuencial)
  - Actualización de las paridades reales, cuando el disco *log* está lleno o pasado un umbral de tiempo

Colocación de la paridad

- Se suele utilizar paridad por cada *stripe*, distribuida entre el conjunto de discos
- Existen diferentes organizaciones:
  - *Right-asymmetric*
  - *Right-symmetric*
  - *Left-asymmetric*
  - *Left-symmetric*
  - *Flat-left-symmetric*

Right-asymmetric vs *right-symmetric*

Right-asymmetric

P0	0	1	2	3
4	P1	5	6	7
8	9	P2	10	11
12	13	14	P3	15
16	17	18	19	P4
P5	20	21	22	23

- Problemas con las lecturas + grandes. Ej: 6-10, no lee de los 5 discos

Right-symmetric

P0	0	1	2	3
7	P1	4	5	6
10	11	P2	8	9
13	14	15	P3	12
16	17	18	19	P4
P5	20	21	22	23

- Problemas mayores aún con las lecturas. Ej: 1-4, no lee de los 4 discos
- Peor que el *right-asymmetric*

Left-asymmetric vs *left-symmetric*

Left-asymmetric

0	1	2	3	P0
4	5	6	P1	7
8	9	P2	10	11
12	P3	13	14	15
P4	16	17	18	19
20	21	22	23	P5

- Problemas con las lecturas de nuevo. Ej: 1-5, no lee de los 5 discos

Left-symmetric

0	1	2	3	P0
5	6	7	P1	4
10	11	P2	8	9
15	P3	12	13	14
P4	16	17	18	19
20	21	22	23	P5

- Haga la petición que haga siempre empleo el número máximo de discos

Flat-left-asymmetric

Flat-left-asymmetric

0	1	2	3	4
5	6	7	8	9
10	11	12	13	14
15	16	17	18	19
20	21	22	23	P5
P4	P3	P2	P1	P0

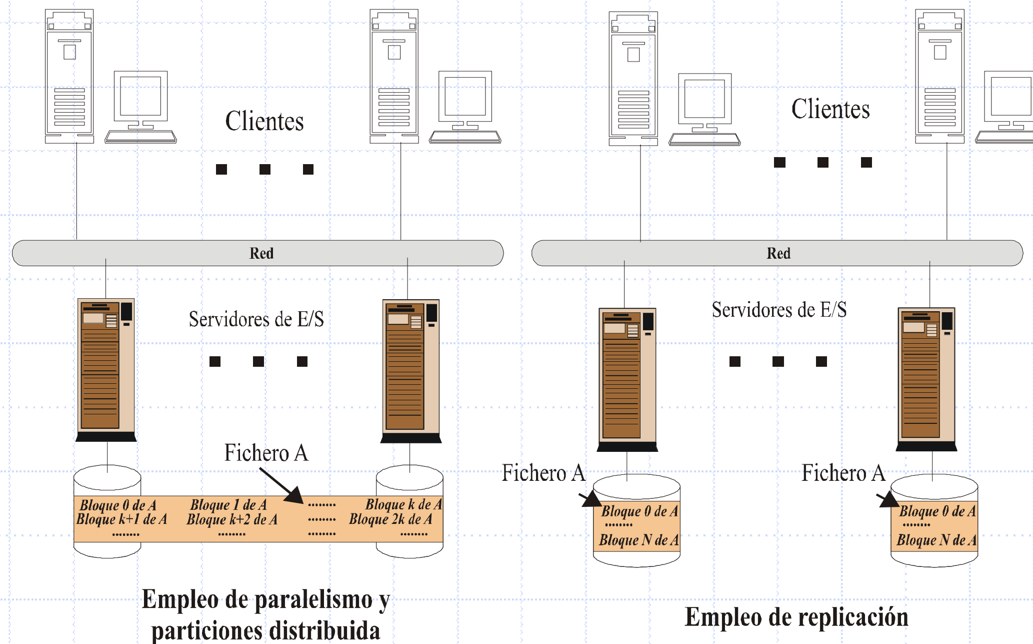
- Funciona muy bien con las lecturas
- Respecto a las escrituras, si utilizas mecanismos para evitar problemas de *small-writes*, no va mal, aunque peor que la lectura
- Se utiliza en sistemas donde haya un mayor porcentaje de lecturas

Índice

- Motivación
- Sistemas RAID
- Bibliotecas de E/S y sistemas de ficheros paralelos
- Técnicas de optimización de la E/S
- Interfaz de E/S. MPI-IO
- Conclusiones

E/S paralela vs E/S replicada

- Un sistema de ficheros replicado almacena una copia completa de cada fichero. En los sistemas de ficheros paralelos cada disco almacena una parte del fichero, de forma que es posible acceder al fichero en paralelo, utilizando menos cantidad de disco y sin problemas de consistencia.



Bibliotecas y sistemas de ficheros paralelos

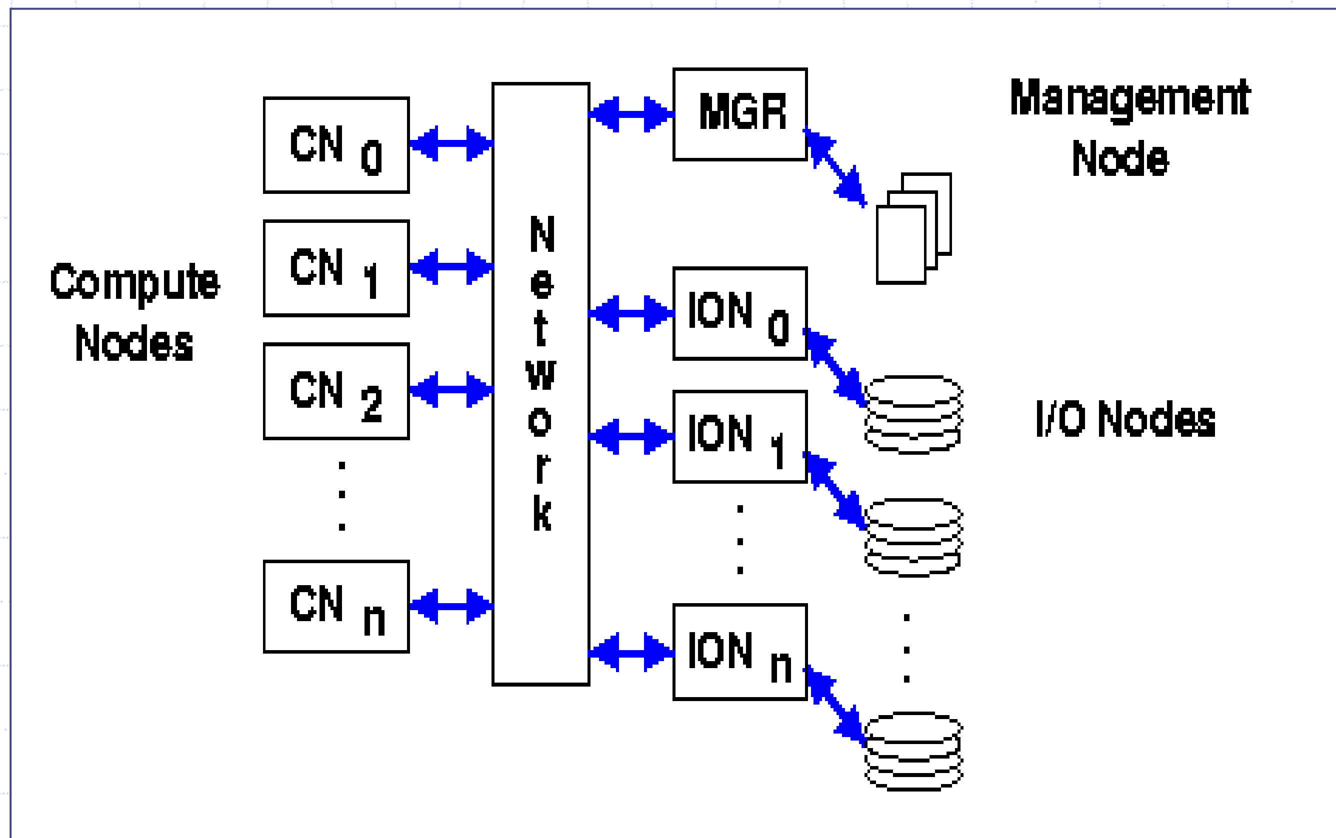
- Bibliotecas de E/S paralela
  - Orientados a un campo específico
    - Ej: ChemIO: *High-Performance I/O for Computational Chemistry Applications*
  - Otras: PASSION, Panda, Jovian, DRA, MIOS, VIP-FS, VIPIOS
  - **MPI-IO**: Iniciativa de estándar!!
- Sistemas de ficheros paralelos
  - nCUBE, Vesta, SFS, PFS, HFS, PIOUS, PPFS, ParFiSys, Galley, PVFS, MFS,…

E/S paralela

- Comerciales:
  - IBM SP2 PIOFS, Intel iPSC CFS, Paragon PFS, Ncube, CMMD
- Investigación:
  - PIOUS, PETSc/Chameleon I/O, PPFS, Vesta, Jovian, PASSION, VIP-FS, Panda, PVFS, MFS

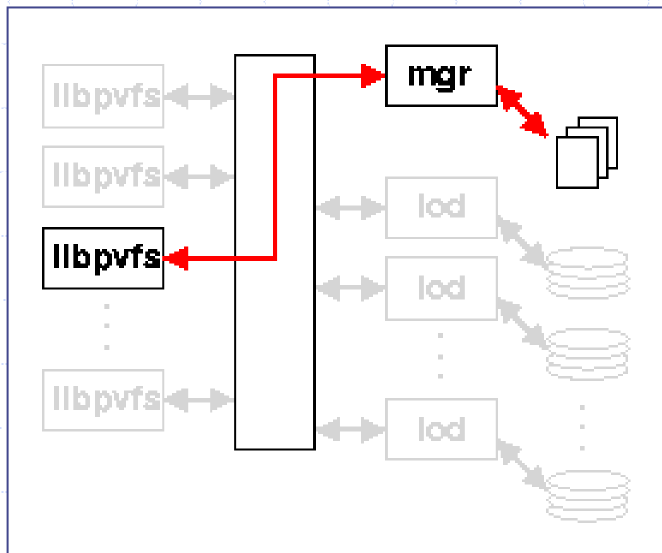
PVFS (*Parallel Virtual File System*)

http://www.parl.clemson.edu/pvfs/

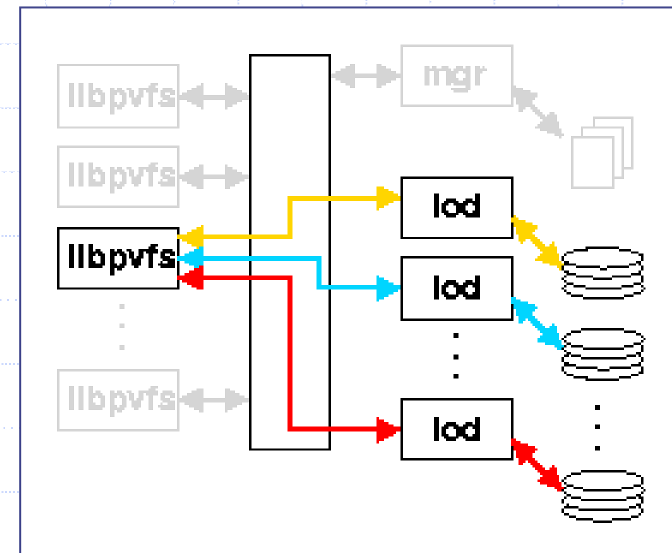


PVFS (Acceso a la información y a la metainformación)

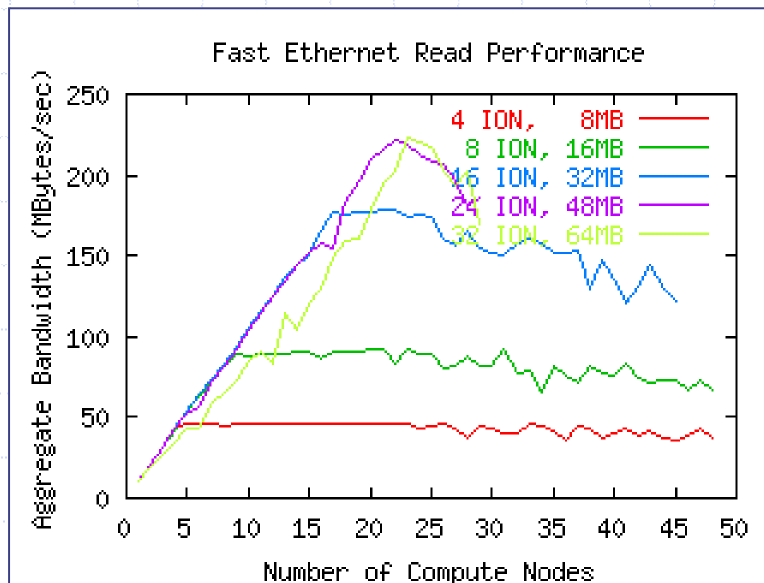
Acceso a la metainformación:

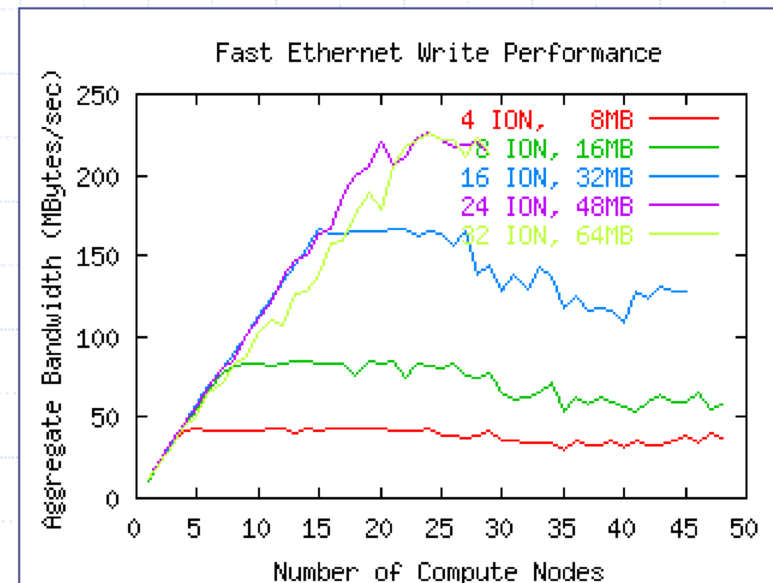


Acceso a los datos:

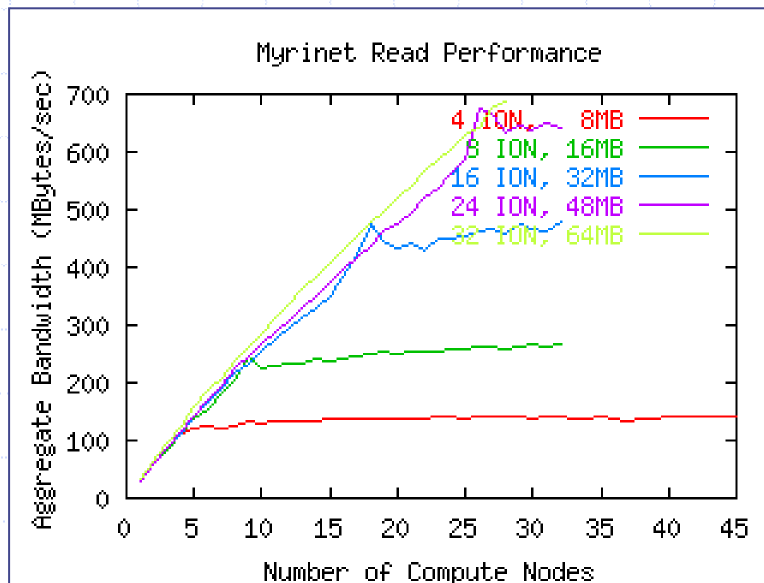


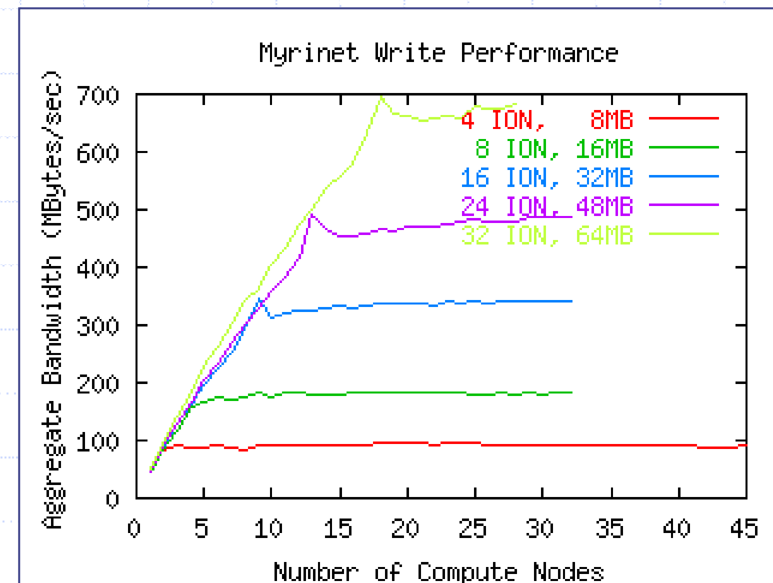
PVFS (Rendimiento con Fast Ethernet)





PVFS (Rendimiento con Myrinet)





Índice

- Motivación
- Sistemas RAID
- Bibliotecas de E/S y sistemas de ficheros paralelos
- Técnicas de optimización de la E/S
- Interfaz de E/S. MPI-IO
- Conclusiones

Técnicas de optimización de la E/S paralela

- Objetivo: reducir la latencia de E/S
- Técnicas:
  - Cribado de datos (*data sieving*)
  - Operaciones colectivas
  - Interfaces de alto nivel
  - Uso de patrones de acceso
  - *Caching* y *prefetching*
  - Utilización de *hints*

Cribado de datos

- Se utiliza para mejorar las peticiones no contiguas de un proceso
- Reducir la latencia de E/S, reduciendo el número de peticiones al sistema de ficheros
  - Pocas peticiones grandes y contiguas
  - Extracción, en memoria, de los datos que realmente se necesitan
  - Se leen más datos de los requeridos
- Ej: ROMIO

Operaciones colectivas

- Se utiliza para mejorar las peticiones no contiguas de múltiples procesos
- Las peticiones de E/S de diferentes procesos se combinan a fin de obtener una única petición de E/S mayor
- Incrementa el ancho de banda de E/S efectivo
- Ej: ROMIO

Interfaces de alto nivel

- Incremento de la funcionalidad de las interfaces tradicionales de operaciones de E/S
  - Ej.: operación para realizar la lectura de una matriz
- Ej: MPI-IO

Patrones de acceso

- Dos aspectos
  - Disposición de los datos (*data layout*) en los ficheros: patrón de almacenamiento (*storage pattern*)
  - Distribución de los datos a través de los nodos: patrón de acceso (*access pattern*)
- La disposición de los datos es crítica en la mejora del rendimiento
- El conocimiento de los patrones de acceso permite utilizar dicha información de la forma más eficiente posible

Patrones de acceso

- nCUBE y Vesta permiten al usuario mayor control sobre el *layout*
- Algunos sistemas de E/S paralela están orientados a datos de un campo específico. Ej.: ChemIO (datos de química)
- Panda oculta los detalles físicos a las aplicaciones
- MPI-IO utiliza tipos de datos MPI (*MPI datatypes*) para describir la disposición de los datos tanto en memoria como en fichero (`MPI_File_set_view()`)

Caching y *prefetching*

- Localidad temporal y espacial de los datos
- Dos tipos de cache:
  - Cache de bloques
  - Cache de metainformación
- Mejora de operaciones de lectura y escritura:
  - Lectura: Uso de *prefetching* (lectura adelantada)
  - Escritura: Uso de escritura diferida
- Utilización de cache genera problemas de coherencia -> Resolución mediante algoritmos de coherencia. Ej.: caches cooperativas

Técnicas de *prefetching* sencillas

- *Sequential read-ahead*: Explota las operaciones de E/S secuenciales.
  - Limitado uso para ficheros pequeños
  - No eficiente para patrones de acceso no secuenciales
- One-block look-ahead
  - Predice bloque i+1
- Infinite-block look-ahead
  - Predice todos los bloques futuros
- Portion-recognition (PORT)
  - Reconoce porciones regulares de datos

Otras técnicas de *prefetching*

- Uso de históricos
- Modelos probabilísticos de las secuencias de peticiones
- *Informed prefetching*
  - Uso de *hints* o *discloses*
  - También existe *informed caching*

Hints

- Objetivo: mejorar el rendimiento del sistema de E/S
- Lampson, 1983
  - Sistemas operativos: Alto, Pilot
  - Redes: Arpanet, Ethernet
  - Lenguajes: Smalltalk
- Inferir accesos futuros basados en accesos pasados
- Expresar conocimiento avanzado sobre un componente del sistema (ej. Sistema de memoria virtual, cache)

Hints

- Tipos de *hints*:
  - *Advising hints*:
    - Recomiendan cómo gestionar los recursos a fin de incrementar el rendimiento
  - *Disclosing hints*:
    - Describen el conocimiento sobre el comportamiento de la aplicación
    - Ventajas:
      - Información independiente de la implementación. Mecanismo de optimización de E/S portable
      - Se pueden seleccionar diferentes políticas
      - Se expresa en términos de la interfaz del sistema

Hints

- Ej.: MPI-IO
- Estructura `MPI_Info`:
  - Pares `(clave, valor) ->` Información adicional sobre las operaciones de E/S
  - Conjunto de claves reservadas
  - Ej.: Clave `striping_unit`, unidad de distribución de los ficheros a través de los dispositivos de E/S

mperez@fi.upm.es

Índice

- Motivación
- Sistemas RAID
- Bibliotecas de E/S y sistemas de ficheros paralelos
- Técnicas de optimización de la E/S
- Interfaz de E/S. MPI-IO
- Conclusiones

Interfaz de los sistemas de E/S paralelos

- Tipos de operaciones:
  - Operaciones básicas
  - Operaciones de lectura y escritura no contigua
  - Operaciones de lectura y escritura no bloqueante
  - Operaciones colectivas
  - Operaciones con punteros compartidos
  - Otras operaciones
- Ej: MPI-IO

MPI-IO

- MPI: *Message Passing Interface*
  - MPI-2:
    - Incluye soporte a la E/S paralela: **MPI-IO** (Una extensión del estándar MPI)
- Objetivo:
  - Proporcionar un estándar para describir operaciones de E/S paralelas dentro de una aplicación de paso de mensajes MPI
    - Escribir en un fichero: enviar un mensaje
    - Leer de un fichero: recibir un mensaje

API de E/S de UNIX

- La mayoría de los sistemas de ficheros paralelos tienen una API estilo UNIX
  - *open:* Abrir un fichero
  - *creat:* Crear un fichero
  - *read:* Leer n bytes desde la posición actual del puntero
  - *write:* Escribir n bytes desde la posición actual del puntero
  - *close:* Cerrar un fichero
  - *lseek:* Mover el puntero del fichero

Otras funciones de E/S

- *readv, writev*
  - Lee/escribe de/a múltiples *buffers* en memoria
  - Se asume que los datos están localizados de forma contigua en el fichero
  - La mayoría de las implementaciones tienen un límite de 16 *buffers* de memoria diferentes para una llamada
  - De uso limitado, porque los usuarios necesitan especificar más frecuentemente no contigüidad en el fichero que en memoria

E/S asíncrona

- *aio_read*, *aio_write* en POSIX
  - No bloqueante: El usuario puede comprobar la finalización de una operación o la implementación puede utilizar una señal
  - La mayoría de las implementaciones tienen un límite en el número máximo de operaciones asíncronas concurrentes; el limite es pequeño, aunque no está predefinido
  - El rendimiento no es suficientemente bueno aún

Lista de operaciones de E/S

- *lio_listio* en POSIX
  - Permite especificar una lista de operaciones de E/S
  - Esta lista puede estar formada por una mezcla de lecturas y escrituras
  - Cada operación se realiza internamente como una operación *aio_read*, *aio_write*
  - Se puede especificar si *lio_listio* es no bloqueante o si termina después de la finalización de todas las peticiones

Desventajas de *lio_listio*

- No trata la lista como una única petición
- No E/S colectiva
- No portable (no está implementada en todos los sistemas)

Patrones de acceso de E/S en aplicaciones paralelas

- Diferentes de los patrones de los programas secuenciales
- Los programas secuenciales acceden normalmente a los datos de forma contigua
- En el caso de varios procesos, cada uno de ellos puede necesitar acceder a zonas no contiguas de datos de un fichero
  - Ej.: bytes 0—10, 20—30, 40—50,...
- Además, grupos de procesos pueden necesitar acceder al fichero simultáneamente, y los accesos de diferentes procesos pueden estar intercalados en el fichero

Problema del API UNIX respecto a la E/S paralela

- No se pueden expresar accesos no contiguos al fichero en una única llamada a función
- Se debe acceder a cada pieza contigua de forma separada. Demasiadas llamadas a sistema y rendimiento bajo
- No E/S colectiva

Historia de MPI-IO

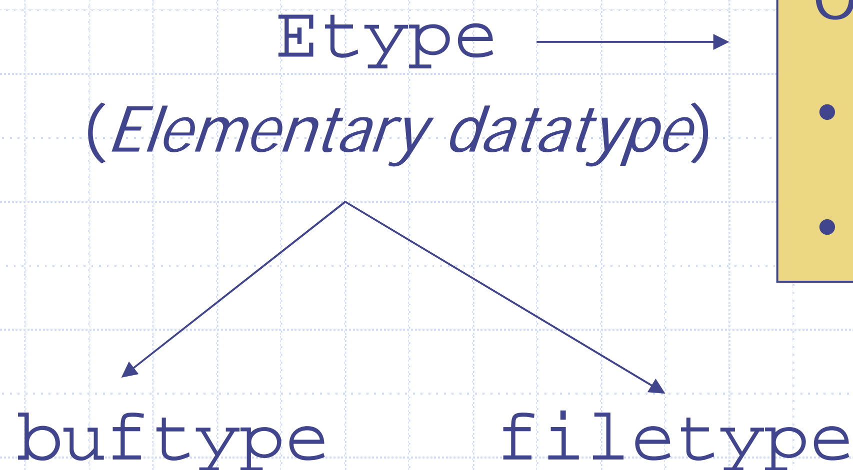
- El artículo de Marc Snir (IBM Watson) explora MPI como contexto para E/S paralela (1994, después MPI-1)
- Grupo de discusión MPI-IO liderado por Jean-Pierre Prost (IBM) y Bill Nitzberg (NASA)
- El grupo MPI-IO se une al Fórum de MPI-2 en Junio de 1996
- El estándar MPI-2 aparece en Julio 1997

Conceptos y características de MPI-IO

- Comunicador:
  - Colección de procesos, los cuales se comunican a través del envío y recepción de mensajes
  - Grupo de procesos que puede acceder a un fichero en operaciones de E/S
- Acceso a fichero:
  - Independiente (no existe coordinación entre los procesos)
  - Colectivo (cada proceso del grupo asociado con el comunicador debe participar en el acceso colectivo)
- Acceso no contiguo tanto en memoria como en fichero
- Tipos de datos MPI
  - Tipos de datos MPI básicos:
    - Aquéllos que corresponden a los tipos de datos básicos en el lenguaje de programación, ej.: MPI_INT, MPI_FLOAT
  - Tipos de datos MPI derivados:
    - Expresar la distribución de los datos en el fichero y el particionado de los datos del fichero entre los procesos

Tipos de datos MPI derivados

- Describen cómo los datos se distribuyen:
  - Los *buffers* de usuario: `buftype`
  - El fichero: `filetype`

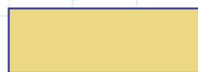


Objetivos:

- Consistencia
- Portabilidad

Filetype

- Define el patrón de los datos que es replicado a través del fichero

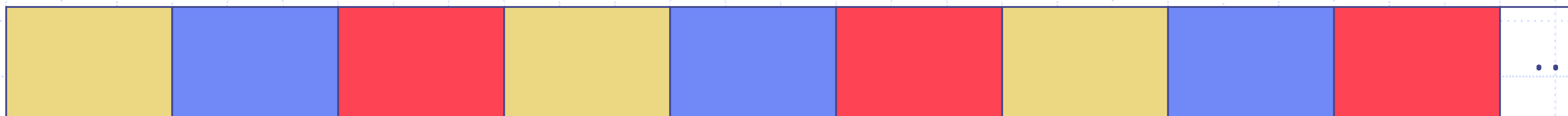


filetype proceso 1
etype proceso 1

filetype proceso 2
etype proceso 2

filetype proceso 3
etype proceso 3

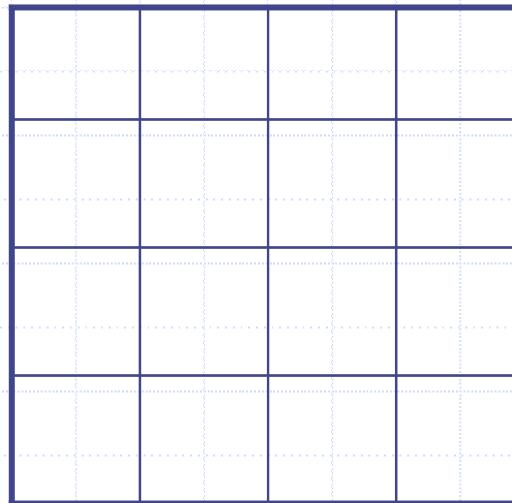
Patrones de distribución comunes:
- broadcast/reduce
- scatter/gather



Buftype

- Describe la distribución en los *buffers* de los usuarios

Buffer proceso



Accesos no contiguos

- Se pueden especificar datos no contiguos en memoria y fichero utilizando tipos de datos MPI, tanto predefinidos como derivados
- La distribución de los datos en **memoria** se especifica en **cada llamada**, como en paso de mensajes
- La distribución de los datos en el **fichero** se define a través de una **vista de fichero**
- Un proceso puede acceder a los datos sólo a través de esta vista
- Se pueden cambiar las vistas

API de MPI-IO

- MPI_File_open();
- MPI_File_close();
- MPI_File_read() y familia;
- MPI_File_write() y familia;
- MPI_File_seek();
- MPI_File_set_view();

MPI_File_open() : Abrir un fichero

- Cabecera:

  *int MPI_File_open(MPI_Comm comm, char *filename, int amode, MPI_Info info, MPI_File *fh);*

- Argumentos:
  - *comm:* comunicador
  - *filename:* nombre del fichero
  - *amode:* modo de acceso al fichero
  - *info:* objeto *info*
  - *fh*: manejador del fichero

- Descripción:
  - Abre el fichero cuyo nombre es *filename*, establece la vista por defecto en el fichero, y establece el modo de acceso *amode.* Se trata de una operación colectiva. *comm* debe ser un intracomunicador válido. Los valores especificados por *amode* deben ser identicos para todos los procesos participantes. El programa es erróneo cuando los procesos participantes no se refieren al mismo fichero a través de sus propias instancias de *filename*.

MPI_File_open() (II)

- *amode*:
  - MPI_MODE_APPEND: Establece como posición inicial de todos los punteros de fichero el final del mismo
  - MPI_MODE_CREATE: Crea el fichero si no existe
  - MPI_MODE_DELETE_ON_CLOSE: Borra el fichero al cerrarlo
  - MPI_MODE_EXCL: genera un error si el fichero ya existe y se utiliza MPI_MODE_CREATE
  - MPI_MODE_RDONLY: sólo lectura
  - MPI_MODE_RDWR: lectura y escritura
  - MPI_MODE_SEQUENTIAL: el fichero sólo se accede de forma secuencial
  - MPI_MODE_UNIQUE_OPEN: el fichero no se abrirá concurrentemente
  - MPI_MODE_WRONLY: sólo escritura

MPI_File_open() (III)

- Ejemplo:

```
MPI_File fh;
MPI_File_open(MPI_COMM_SELF,
    "test.txt",
    MPI_MODE_CREATE|MPI_MODE_RDWR,
    MPI_INFO_NULL, &fh);
```

MPI_File_close() : Cerrar un fichero

- Cabecera:

  *int MPI_File_close (MPI_File *fh);*

- Argumentos:
  - fh: manejador del fichero
- Descripción:
  - Cierra el fichero manejado por *fh* y libera las estructuras de datos internas asociadas. Se trata de una operación colectiva. El fichero también se borra si se estableció el *flag* MPI_MODE_DELETE_ON_CLOSE cuando se abrió el fichero. En esta situación, si otras tareas tenían abierto el fichero y están accediendo a él de forma concurrente, los accesos se realizarán normalmente, como si el fichero no se hubiera borrado, hasta que dichas tareas cierren el fichero.
- Ejemplo:

  MPI_File_close(&fh);

MPI_File_read() : Leer de un fichero

- Cabecera:

  *int MPI_File_read (MPI_File fh, void *buf, int count, MPI_Datatype datatype,MPI_Status *status);*

- Argumentos:
  - fh: manejador del fichero
  - buf: la dirección inicial del *buffer*
  - count: número de elementos del *buffer*
  - datatype: el tipo de datos de cada elemento del *buffer*
  - status: estado de la operación

- Descripción:

  Intenta leer del fichero manejado por *fh*, *count* elementos de tipo *datatype* en el *buffer buf*, comenzando por la posición del puntero de fichero

- Ejemplo:
  - *MPI_File_read(fh, buf, count, MPI_INT, &status);*

MPI_File_read _all() : Lectura colectiva

- **Cabecera:**

  *int MPI_File_read_all (MPI_File fh, void *buf, int count, MPI_Datatype datatype,MPI_Status *status);*

- **Argumentos:**
  - fh: manejador de fichero
  - buf: la dirección inicial del *buffer*
  - count: número de elementos del *buffer*
  - datatype: el tipo de datos de cada elemento del *buffer*
  - status: estado de la operación

- **Descripción:**

  Versión colectiva de *MPI_File_read()*

- **Ejemplo:**
  - *MPI_File_read_all(fh, buf, count, MPI_INT, &status);*

mperez@fi.upm.es

MPI_File_iread() : Lectura no bloqueante

- Cabecera:

  *int MPI_File_iread (MPI_File fh, void *buf, int count, MPI_Datatype datatype,MPI_Request *request);*

- Argumentos:
  - fh: manejador de fichero
  - buf: dirección inicial del buffer.
  - count: número de elementos del *buffer*
  - datatype: el tipo de datos de cada elemento de *buffer*
  - request: petición de estado

- Descripción:

  Esta rutina es la versión no bloqueante de MPI_File_read(). Realiza la misma función que MPI_File_read(), salvo que retorna inmediatamente y almacena un enlace a un objeto de tipo *request*.

- Ejemplo:
  - *MPI_File_iread(fh, buf, count, MPI_INT, &request);*

MPI_Test() y MPI_Wait()

- *int MPI_Test (MPI_Request *request, int *flag, MPI_Status *status);*
- *int MPI_Wait (MPI_Request *request, MPI_Status *status);*
- Descripción:
  - MPI_Test() devuelve *flag* = **true** si la operación identificada por *request* está completada. El objeto *status* contendrá información sobre el estado de finalización de la operación.
  - MPI_Wait() retorna después de que la operación identificada por **request** se ha completado. Si el objeto asociado con **request** se creó mediante una operación no bloqueante, el objeto es liberado y **request** se establece a MPI_REQUEST_NULL.

MPI_File_read_shared() : Puntero de fichero compartido

- **Cabecera:**

  *int MPI_File_read_shared (MPI_File fh, void *buf, int count, MPI_Datatype datatype,MPI_Status *status);*

- **Argumentos:**
  - fh: manejador de fichero
  - buf: dirección inicial del *buffer*.
  - count: número de elementos del *buffer*
  - datatype: el tipo de datos de cada elemento del *buffer*
  - status: estado de la operación

- **Descripción:**

  Esta rutina intenta leer del fichero manejado por *fh*, *count* elementos de tipo *datatype* en el *buffer buf*, comenzando por la posición del fichero determinado por el valor del puntero compartido.

- **Ejemplo:**
  - *MPI_File_read_shared(fh, buf, count, MPI_INT, &status);*

MPI_File_write() : Escribir a un fichero

- Cabecera:

  *int MPI_File_write (MPI_File fh, void *buf, int count, MPI_Datatype datatype,MPI_Status *status);*

- Argumentos:
  - fh: manejador del fichero
  - buf: dirección inicial del *buffer*
  - count: número de elementos del *buffer*
  - datatype: el tipo de datos de cada elemento del *buffer*
  - status: estado de la operación

- Descripción:

  Esta rutina intenta escribir en un fichero manejado por *fh*, *count* elementos de tipo *datatype* del *buffer buf*, comenzando en la posición determinado por el valor del puntero individual del fichero.

- Ejemplo:
  - *MPI_File_write(fh, buf, count, MPI_INT, &status);*

MPI_File_write _all() : Escritura colectiva

- **Cabecera:**

  *int MPI_File_write_all (MPI_File fh, void *buf, int count, MPI_Datatype datatype,MPI_Status *status);*

- **Argumentos:**
  - fh: manejador del fichero
  - buf: dirección inicial del *buffer*
  - count: número de elementos del *buffer*
  - datatype: tipo de datos de cada elemento del *buffer*
  - status: estado de la operación

- **Descripción:**

  Versión colectiva de MPI_File_write()

- **Ejemplo:**
  - *MPI_File_write_all(fh, buf, count, MPI_INT, &status);*

mperez@fi.upm.es

MPI_File_iwrite() : Escritura no bloqueante

- **Cabecera:**

  *int MPI_File_iwrite (MPI_File fh, void *buf, int count, MPI_Datatype datatype,MPI_Request *request);*

- **Argumentos:**
  - fh: manejador del fichero
  - buf: dirección inicial del *buffer*
  - count: número de elementos del *buffer*
  - datatype: tipo de datos de cada elemento del *buffer*
  - request: petición de estado

- **Descripción:**

  Versión no bloqueante de MPI_File_write(). Realiza la misma función que MPI_File_write(), salvo que retorna inmediantamente y almacena un enlace a un objeto de tipo *request*.

- **Ejemplo:**
  - *MPI_File_iwrite(fh, buf, count, MPI_INT, &request);*

MPI_File_write_shared() : Puntero de fichero compartido

- Cabecera:

  *int MPI_File_write_shared (MPI_File fh, void *buf, int count, MPI_Datatype datatype,MPI_Status *status);*

- Argumentos:
  - fh: Manejador de fichero
  - buf: Dirección inicial del *buffer*
  - count: Número de elementos del *buffer*
  - datatype: Tipo de datos de cada elemento del *buffer*
  - status: estado de la operación

- Descripción:

  Esta rutina intenta escribir en el fichero manejado por *fh*, *count* elementos de tipo *datatype* del *buffer buf*, comenzando por la posición del fichero determinado por el valor del puntero compartido.

- Ejemplo:
  - *MPI_File_write_shared(fh, buf, count, MPI_INT, &status);*

Funciones de acceso a los datos MPI-IO

Posicionamiento	Sincronización	Coordinación	
		Independiente	*Colectivo*
Desplazamientos explícitos	Bloqueante	MPI_File_read_at MPI_File_write_at	MPIO_read_at_all MPIO_write_at_all
	No bloqueante	MPI_File_iread_at MPI_File_iwrite_at	
Punteros de fichero individuales	Bloqueante	MPI_File_read MPI_File_write	MPI_File_read_all MPI_File_write_all
	No bloqueante	MPI_File_iread MPI_File_iread	
Punteros de fichero compartidos	Bloqueante	MPI_File_read_shared MPI_File_write_shared	
	No bloqueante	MPI_File_iread_shared MPI_File_iwrite_shared	

Posicionamiento

- Dos conceptos para describir ubicaciones de un fichero:
  - Tipos de datos MPI
  - Desplazamientos
    - new_file_position=old_position+(size(buftype)*bufcount)/size(etype)
- El puntero de fichero se actualiza al comienzo de cada acceso con el número de datos solicitado

Sincronización

- El uso de funciones de acceso de datos no bloqueantes puede mejorar el rendimiento, a través del solapamiento de las fases de computación y E/S.
- Una llamada (MPI_Wait o MPI_Test) se necesita para completar las peticiones de E/S o certificar que los datos han sido leídos/escritos.
- La única forma de garantizar que los datos se han escrito realmente es utilizando la llamada MPIO_File_sync.

Coordinación

- Operaciones colectivas: "Todos" los procesos en el grupo comunicador que abrió el fichero deben participar en la operación
- Diferencias independiente-colectivo:
  - Punto de vista semántico: Sincronización potencial
  - Punto de vista del rendimiento: La versión colectiva puede ser mucho más rápida

MPI_File_seek() : Mover el puntero de fichero

- Cabecera:

  int MPI_File_seek (MPI_File fh,MPI_Offset offset, int whence);

- Argumentos:
  - fh: manejador del fichero
  - offset: desplazamiento del fichero
  - whence: base del desplazamiento
- Descripción:
  - Actualiza el puntero de fichero individual de acuerdo a *whence*.
- Valores *whence*:
  - **MPI_SEEK_CUR:** el nuevo puntero es igual al actual más *offset*
  - **MPI_SEEK_END:** el nuevo puntero es igual al final del fichero más *offset*
  - **MPI_SEEK_SET:** el nuevo puntero es igual a *offset*
- Ejemplo:
  - *MPI_File_seek(fh, -10, MPI_SEEK_CUR);*
  - *MPI_File_seek(fh, 0, MPI_SEEK_SET);*

MPI_File_set_view() : Establecer una vista del fichero

- Cabecera:

  *int MPI_File_set_view (MPI_File fh, MPI_Offset disp, MPI_Datatype etype, MPI_Datatype filetype, char *datarep, MPI_Info info);*

- Argumentos:
  - fh: manejador del fichero
  - disp: desplazamiento
  - etype: tipo de datos elemental
  - filetype: *filetype*
  - datarep: representación de los datos
  - info: objeto **info**

- Descripción:
  - Asocia una nueva vista definida por **disp**, **etype**, **filetype**, y **datarep** con el fichero abierto referenciado mediante *fh*. Se trata de una operación colectiva. Todos los procesos participantes deben especificar los mismos valores para *datarep* y *etype*.

MPI_File_set_view() (II)

- Valores *datarep*:
  - **external32:** Establece que las operaciones de lectura y escritura se convierten en la representación **external32**, documentada en el estándar MPI-2.
  - **Internal:** Puede utilizarse para operaciones de E/S en un entorno homogéneo o heterogéneo. IBM ha definido su formato interno con la intención de que cualquier implementación de MPI proporcionada por IBM pueda utilizarlo.
  - **native:** Debe utilizarse en la mayoría de las situaciones. Los datos se almacenan en el fichero exactamente igual que en memoria. Esta representación es adecuada en un entorno homogéneo MPI y no implica ningún coste de conversión.
- Ejemplo:
  - *MPI_File_set_view(fh, 0, MPI_INT, MPI_INT, "native", MPI_INFO_NULL);*

Ejemplo (Matriz 2-D)

Estructura del fichero:

1	2	3
4	5	6
7	8	9

Distribuir
y
trasponer

1	4	7
2	5	8
3	6	9

Buffer proceso 1

Buffer process 2

Buffer proceso *3*

Ejemplo (Matriz 2-D) (II)

Implementación utilizando *filetypes* y *buftypes*:

Filetype proceso 1:

Filetype proceso 2:

Filetype proceso 3 :

layout real en el fichero:

buftype (todos los procesos):

Análisis (matriz 2-D)

Estructura de fichero:

rowtype

1	2	3
4	5	6
7	8	9

Distribuir
y
transponer

columntype

1	4	7
2	5	8
3	6	9

Buffer proceso 1

Buffer proceso 2

Buffer proceso 3

ftype = set of rowtype

buftype = set of columntype

Ejemplo (Matriz 2-D) (Cód. I)

```
read_matrix(
    char *fname, /* File containing matrix "A[n][n]" */
    int n, /* Number of rows (columns) of matrix */
    MPI_Datatype etype, /* Matrix element type */
    void *localA) /* Target for transposed matrix */
{
    MPI_File fh;
    MPI_Datatype ftype, buftype;
    MPI_Status status;
    MPI_Datatype column_t, rowtype;
    int m, rank, nrows, sizeofetype, sizeofrowtype;
```

Ejemplo (Matriz 2-D) (Cód. II)

```
/* Create row-cyclic filetype for data distribution */
    MPI_Type_size(etype, &sizeofetype);
    MPI_Type_hvector(n,1, sizeofetype, etype,&rowtype);
    MPI_Type_size(rowtype, &sizeofrowtype);
    MPI_Type_hvector(n,1, n*sizeofrowtype, rowtype, &ftype);
    MPI_Type_commit(&ftype);
    MPI_Type_free(&rowtype);
/* Create buftype to transpose matrix into process
    memory */
    MPI_Comm_size(MPI_COMM_WORLD, &m);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Type_hvector(n,1, n*sizeofetype, etype, &column_t);
    MPI_Type_hvector(n, 1, sizeofetype, column_t, &buftype);
    MPI_Type_commit(&buftype);
    MPI_Type_free(&column_t);
```

Ejemplo (Matriz 2-D) (Cód. III)

```
/* Read, distribute and transpose the matrix (and cleanup) */
MPI_File_open(MPI_COMM_WORLD, fname,
              MPI_MODE_RDONLY, MPI_INFO_NULL, &fh);
MPI_File_set_view(fh, rank*n*sizeofetype, etype, ftype,
                  "native", MPI_INFO_NULL);
MPI_File_read_all(fh, localA, 1, buftype, &status);
MPI_File_close(&fh);
MPI_Type_free(&ftype);
MPI_Type_free(&buftype);
}
```

Ejemplo (Matriz 2-D) (Cód. IV)

```
int main(int argc, char *argv[]) {
        char buffer[20];
        int i, j, m, rank;

        if (argc != 2 ) {
                fprintf(stderr,"Error. Usage: matrix filename \n");
                exit(1);
        }
        MPI_Init(&argc, &argv);
        MPI_Comm_size(MPI_COMM_WORLD, &m);
        MPI_Comm_rank(MPI_COMM_WORLD, &rank);
        read_matrix(argv[1],m,MPI_CHAR,buffer);
```

Ejemplo (Matriz 2-D) (Cód. V)

```
sleep(rank*5);
printf("The transposed matrix of the process
        %d is:\n",rank);
for (i=0;i<m;i++) {
        for (j=0;j<m;j++){
                printf("%c ",buffer[i*m+j]);
        }
        printf("\n");
}
MPI_Finalize();
exit(0);
}
```

Ejemplo (Matriz 2-D)

Fichero de datos:

1234abcd$%&/zyxw5678efgh$
%&/zyxw1234ijkl$%&/zyxw567
8mnop$%&/zyxw

Ejemplo (Matriz 2-D)

```
[mperez@yedra romio]# mpirun -c 4 matrix hola4
The transposed matrix of the process 0 is:
1 5 1 5
2 6 2 6
3 7 3 7
4 8 4 8
The transposed matrix of the process 1 is:
a e i m
b f j n
c g k o
d h l p
The transposed matrix of the process 2 is:
$ $ $ $
% % % %
& & & &
/ / / /
The transposed matrix of the process 3 is:
z z z z
y y y y
x x x x
w w w w
[mperez@yedra romio]#
[mperez@yedra romio]#
[mperez@yedra romio]#
```

Optimizaciones MPI-IO

- *Data sieving*
- E/S colectiva
- *Prefetching* y *caching* mejorados

Implementaciones MPI

- MPICH
  - http://www-unix.mcs.anl.gov/mpi/mpich/
- LAM MPI
  - http://www.lam-mpi.org/
- WinMPI
  - http://csalpha.ist.unomaha.edu/~rewini/WinMPI/
- ROMIO
  - http://www-unix.mcs.anl.gov/romio/

Índice

- Motivación
- Sistemas RAID
- Bibliotecas de E/S y sistemas de ficheros paralelos
- Técnicas de optimización de la E/S
- Arquitecturas de red
- Interfaz de E/S. MPI-IO
- Conclusiones

Conclusiones

- La E/S Paralela intenta resolver la crisis de E/S
- MPI-IO proporciona un estándar para describir operaciones de E/S paralelas dentro de aplicaciones MPI.
- Esta interfaz no es obvia!!!
- Si se requiere un entorno heterogéneo y geográficamente disperso, se puede hacer uso de soluciones grid (Data Grid)

Líneas de trabajo futuras

- Desarrollar interfaces de E/S fáciles de usar
- Utilizar otras tecnologías a fin de incrementar el rendimiento de la fase de E/S (por ejemplo, MAPFS (MultiAgent Parallel File System))
- Utilizar *autonomic computing* en los sistemas de E/S para la configuración de los mismos (*self-configuration*).
- Proporcionar características propias de la E/S paralela a los data grids.

Para saber más....

- **Parallel I/O Archive** *(http://www.cs.dartmouth.edu/pario/)*
- **MPI Forum** (*http://www.mpi-forum.org*)
- **The Message Passing Interface (MPI) standard** (http://www-unix.mcs.anl.gov/mpi/)
- **MPI: Message Passing Interface** (http://www.erc.msstate.edu/misc/mpi/)
- Peter Corbett et al. **"Overview of the MPI-IO Parallel I/O Interface"** *Proceedings of the Third Workshop on I/O in Parallel and Distributed Systems, IPPS '95, Santa Barbara, CA.* April 1995.

Para saber más...

- [Nitzberg92] Bill Nitzberg. **"Performance of the iPSC/860 Concurrent File System"** *Technical Report RND-92-020.* December 1992.
- [Rosario93] Juan Miguel del Rosario et al. **"Improved Parallel I/O via a Two-phase Run-time Access Strategy"** *ACM Computer Architecture News. Volume 21(5), pages 31-38.* December 1993.
- [Kotz94] David Kotz. **"Disk-directed I/O for MIMD multiprocessors"** *Proceedings of the 1994 Symposium on Operating Systems Design and Implementation.* November 1994.
- [Seamons95] Kent E. Seamons et al. **"Server-directed collective I/O in Panda"** *Proceedings of Supercomputing '95, San Diego, CA.* December 1995

Para saber más...

- **MPICH** (http://www-unix.mcs.anl.gov/mpi/mpich/)
- **LAM MPI** (http://www.lam-mpi.org/)
- **WinMPI** (http://csalpha.ist.unomaha.edu/~rewini/WinMPI/)
- **ROMIO: A High-Performance, Portable MPI-IO Implementation** (http://www-unix.mcs.anl.gov/romio/)

E/S Paralela

MPI-IO: Un caso de estudio

María S. Pérez (mperez@fi.upm.es)
Dep. Arquit. y Tecnol. de Sist. Informáticos
Facultad de Informática, UPM, Madrid