Análisis de métodos de redistribución de datos para aplicaciones MPI maleables

Iker Martín-Álvarez[1], José I. Aliaga[1], Maribel Castillo[1], Sergio Iserte[2]

Resumen— **La maleabilidad de procesos puede definirse como la capacidad de un trabajo paralelo MPI distribuido para modificar el número de procesos sin detener su ejecución, reasignando los recursos computacionales inicialmente asignados al trabajo tantas veces como sea necesario. En general, la maleabilidad se compone de cuatro etapas: reasignación de recursos, gestión de procesos, redistribución de datos y reanudación de la ejecución. Entre ellas, la tercera etapa es la que más tiempo consume y domina el tiempo total de reconfiguración. En este artículo se comparan diferentes implementaciones de esta etapa utilizando operaciones MPI punto a punto y colectivas, incluyendo sus versiones no bloqueantes, tanto para Ethernet 10G como para Infiniband EDR. Estas estrategias de redistribución de datos se combinan con diferentes métodos para expandir/reducir trabajos utilizando una aplicación que solo realiza la segunda y tercera etapa de la maleabilidad, lo que permite evaluar el coste de las diferentes metodologías de modo aislado. Los resultados muestran que la versión punto a punto no bloqueante junto al método de creación de procesos *Merge*, es la alternativa que más reduce el tiempo de redistribución.**

Palabras clave— **MPI, Maleabilidad, Redistribución de datos, Reconfiguración, Paralelismo**

I. Introducción

La maleabilidad es la técnica que permite modificar en tiempo de ejecución los recursos computacionales asignados a una aplicación. La puesta en práctica de esta técnica en un sistema puede ser muy diversa y puede abordar la asignación de diferentes tipos de recursos. En este trabajo, la maleabilidad se refiere a la capacidad de un trabajo paralelo y distribuido de redimensionar su tamaño, en términos de procesos MPI[1], modificando los recursos computacionales que tenía asignados en cualquier punto de la ejecución, y tantas veces como sea necesario. En los grandes sistemas, es habitual encontrar trabajos de este tipo que utilizan la interfaz de paso de mensajes MPI, como herramienta de desarrollo estándar de facto. Estos trabajos se pueden modificar para que en un momento dado reciban una comunicación del gestor de recursos del sistema (RMS) para modificar su asignación de recursos, con el objetivo de maximizar las prestaciones del sistema de acuerdo a algún criterio definido en su política de gestión.

El uso de la maleabilidad en las aplicaciones se justifica por los beneficios que produce, y puede analizarse de dos formas diferentes. Desde el punto de vista de una aplicación, el beneficio se suele conseguir con la mejora de su rendimiento, que habitualmente se alcanza al expandir el trabajo, es decir, al aumentar el número de recursos (procesadores) asignados para la ejecución. Desde el punto de vista del sistema, este beneficio se obtiene al aumentar el rendimiento global, medido en términos de productividad y trabajos finalizados por segundo [2], [3], [4], [5], [6].

La inclusión de la maleabilidad en un trabajo paralelo suele aprovechar la existencia de puntos de control específicos a lo largo de la ejecución, en los que los procesos se sincronizan, para su activación. En aplicaciones iterativas, estos puntos de control suelen aparecer al inicio de cada iteración, mientras que en aplicaciones no iterativas los puntos se definen al inicio de cada fase.

Una fase de maleabilidad conlleva la realización de varias etapas:

1. *Etapa* 1*: Reasignación de recursos.* Preguntar al RMS si la aplicación tiene que ser reconfigurada, bien porque necesita más recursos o bien porque puede liberar alguno. Esta decisión la tomará el RMS en función de los recursos disponibles y de las necesidades que tengan los trabajos que están en ejecución y en espera en ese momento.
2. *Etapa* 2*: Gestión de procesos.* Crear/finalizar procesos MPI en función de la decisión de reconfiguración tomada por el RMS. A partir de este momento, los procesos originales de un trabajo redimensionado serán considerados como los procesos padre, mientras que los procesos en ejecución tras la finalización de la maleabilidad se identificarán como procesos hijo.
3. *Etapa* 3*: Redistribución de datos.* Comunicar datos entre los procesos padre (NP) e hijo (NH), para que la ejecución pueda continuar correctamente en el mismo punto donde se concluyó el redimensionado, pero utilizando los procesos hijo.
4. *Etapa* 4*: Continuación de la ejecución.* Continuar la ejecución de la aplicación con los hijos.

La primera fase es la que inicia la maleabilidad, por lo que las tres últimas etapas sólo se llevan a cabo si el RMS toma la decisión de reconfigurar la aplicación.

La realización de las etapas 2 y 3, suele suponer un importante sobrecoste que puede afectar negativamente a las prestaciones, tanto de la aplicación como del sistema en su conjunto, por lo que una implementación eficiente resulta fundamental. Además, ambas etapas están muy relacionadas por lo que no se pueden implementar y analizar de forma separada. En [7] se propusieron diferentes métodos y estrategias para llevar a cabo la etapa 2, analizando las pres-

[1]Dpto. de Ingeniería y Ciencia de los Computadores, Universitat Jaume I de Castelló, e-mails: martini@uji.es, aliaga@uji.es, castillo@uji.es

[2]Computer Science Department, Barcelona Supercomputing Center, e-mail: sergio.iserte@bsc.es

taciones y el sobrecoste que cada alternativa suponía tanto para la aplicación como para el sistema en su totalidad. El objetivo del presente trabajo es analizar en detalle las operaciones necesarias para completar la redistribución de datos (etapa 3 de maleabilidad) y evaluar que alternativas son más adecuadas en un escenario aislado. Esto nos llevará a seleccionar que método es más adecuado y, en función del tipo de gestión de procesos que se utilice para llevar a cabo la etapa 2, implementar de forma eficiente las etapas 2 y 3 de la maleabilidad que son fundamentales en esta operación.

Las comunicaciones requeridas para completar una distribución de datos se puede realizar utilizando diferentes operaciones MPI: operaciones punto a punto (P2P), que se realizan entre dos procesos concretos, u operaciones colectivas (COL), en las que están involucrados todos los procesos de un comunicador. Además, cada una de estas operaciones tiene su variante bloqueante y no bloqueante. Estas últimas se utilizan para solapar cálculo y comunicaciones.

En el campo de la maleabilidad, éstas se pueden utilizar para solapar la redistribución de datos con la ejecución de la aplicación maleable, dando lugar a dos alternativas para completar la etapa 3: síncrona o asíncrona. En este trabajo se considera que una operación de comunicación es síncrona cuando el usuario tiene que esperar a que termine. En cambio será asíncrona si el usuario puede realizar otras tareas al mismo tiempo. Por tanto las comunicaciones síncronas están basadas en operaciones bloqueantes, mientras que las asíncronas en no bloqueantes.

En la redistribución síncrona, los procesos padre detienen su ejecución mientras se lleva a cabo esta operación. Mientras que en la redistribución asíncrona, los datos se comunican desde los procesos padre a los hijos como una tarea en segundo plano, sin detener la ejecución de los padres.

También es importante comentar que el rendimiento de las diferentes operaciones de comunicación está muy determinada por la red que se utilice, que influye directamente sobre las diferentes alternativas definidas para completar la redistribución. En este trabajo se compara el funcionamiento de las diferentes alternativas en una red Ethernet 10G y en una red Infiniband EDR, con el fin de mostrar cual de ellas realiza la redistribución en el menor tiempo posible de forma aislada.

El resto del artículo se organiza como sigue. La Sección II describe las principales operaciones MPI requeridas para llevar a cabo una redistribución de datos y explica brevemente diferentes alternativas que se han implementado para realizar la etapa 2 de la maleabilidad. La Sección III describe los métodos utilizados para completar la etapa 3. En la Sección IV se muestran los resultados experimentales obtenidos de aplicar conjuntamente las etapas 2 y 3 de maleabilidad sobre las dos redes de comunicación comentadas sobren un clúster con 8 nodos. Finalmente, la Sección V presenta las conclusiones de este estudio.

II. Operaciones MPI para maleabilidad

La tecnología más utilizada para la implementación de aplicaciones maleables es la Interfaz de Paso de Mensajes (MPI) [1]. Esta define una especificación para el paso de mensajes entre procesos implementada a través de una librería para diversos lenguajes y constituye una tecnología ampliamente utilizada para la paralelización de códigos.

MPI cuenta con una serie de operaciones con las que implementar cualquier comunicación entre procesos, que se pueden dividir en dos tipos: operaciones *punto a punto* y operaciones *colectivas*. Las primeras definen comunicaciones entre dos procesos concretos, mientras que las segundas involucran a todos los procesos de un comunicador. Todo comunicador está definido por un grupo de procesos, en los que cada uno de sus miembros tiene un identificador (*rank*). Si una comunicación se realiza entre procesos de un grupo, el comunicador utilizado se caracteriza como *intra-comunicador*. En cambio, cuando se requiere comunicar entre procesos de dos grupos distintos, es necesario crear un *inter-comunicador*, en los que aparecen procesos de ambos grupos.

A continuación se introducen las operaciones MPI que serán utilizadas para completar las etapas 2 y 3 de la maleabilidad. En primer lugar se introducen las alternativas descritas en [7] para modificar el número de procesos de una aplicación (etapa 2), y posteriormente se describen las operaciones MPI utilizadas para realizar la redistribución de datos (etapa 3).

A. Modificación del número de procesos

En [7], se presentaron diferentes métodos y estrategias con el objetivo de analizar y comparar el rendimiento para expandir o reducir el número de procesos de una aplicación en ejecución, sin desviarse del estándar MPI [1]. A continuación, se describen, de forma muy resumida, los más representativos para que la descripción posterior de los métodos de redistribución de datos que se proponen sean fácilmente comprensibles. Como ya se ha comentado, los métodos aplicados en las etapas 2 y 3 de maleabilidad no pueden analizarse de forma totalmente independiente, ya que en la etapa 2 se generan un número de procesos y un tipo de comunicador que serán utilizados en la etapa 3.

La creación de procesos en MPI se basa en la función `MPI_Comm_spawn` que es una operación colectiva sobre un determinado comunicador. En el Listado 1 se muestra la definición de este rutina, tal y como aparece en el estándar [1].

Listado 1: Definición de la función de MPI para la creación de procesos dinámicos.

```
1 int MPI_Comm_spawn(const char *command,
2   char *argv[], int maxprocs, MPI_Info info,
3   int root, MPI_Comm comm, MPI_Comm *intercomm,
4   int array_of_errcodes[])
```

La alternativa más sencilla para completar la Etapa 2 de la maleabilidad es el uso de la rutina `MPI_Comm_spawn` para generar los NH procesos necesarios para continuar la ejecución. Este método se ha

denominado *Baseline* en [7] y aparece gráficamente en la Figura 1a. En este ejemplo, inicialmente había $NP = 2$ procesos (padres) ejecutando la aplicación, tras completar la llamada a esta función MPI, se crean $NH = 4$ procesos nuevos (hijos). Tras esta llamada, los hijos deben continuar la ejecución de la aplicación en el mismo punto donde la dejaron los padres, y estos deben finalizar su ejecución. El uso del método *Baseline* involucra 3 comunicadores diferentes:

- Intra-comunicador `MPI_COMM_WORLD` definido por todos los padres.
- Intra-comunicador `MPI_COMM_WORLD` definido por todos los hijos.
- Inter-comunicador que conecta los padres con los hijos.

De estos, únicamente el último es necesario para completar la Etapa 3 de la maleabilidad.

El principal problema del método *Baseline* es la suscripción en exceso (*oversubscription*), es decir, que un procesador deba manejar un número de procesos mayor que el número de núcleos que incluya, lo que puede afectar a las prestaciones del sistema. Una posible alternativa es reutilizar los procesos padres existentes como hijos y generar únicamente el resto de procesos necesarios para completar la reconfiguración hasta NH hijos. Este método, denominado *Merge* en [7] se puede ver gráficamente en las Figuras 1d y 1e y las funciones colectivas adicionales en el Listado 2. En el primer caso, reconfiguración desde $NP = 2$ a $NH = 4$, se crean únicamente 2 nuevos procesos, siendo necesario utilizar la rutina `MPI_Intercomm_merge` para generar el intra-comunicador que agrupa a todos los procesos que continuarán la ejecución [8]. Se trata de la primera función del Listado 2, una operación colectiva en la que participan todos los padres e hijos, donde el parámetro `high` fija como se numerarán los procesos en la agrupación final.

Por su parte, en la Figura 1e (reconfiguración desde $NP = 4$ a $NH = 2$), se muestra que cuando $(NP > NH)$ no hace falta crear nuevos procesos, sino que la alternativa es detener la ejecución de $(NP - NH)$ procesos padre. Para ello, se divide el intra-comunicador de los padres en dos comunicadores, utilizando la segunda función del Listado 2, `MPI_Comm_split`. El parámetro `color` se utiliza para obtener varios comunicadores, aunque en este caso solo hace falta dividir el comunicador original en dos partes. Posteriormente, se debe suspender la ejecución de los procesos que no son hijos y continuar la ejecución de la aplicación.

Listado 2: Definición de la funciones de MPI adicionales para la creación de procesos dinámicos con el método *Merge*.

```
1 int MPI_Intercomm_merge(MPI_Comm intercomm,
2   int high, MPI_Comm *newintracomm);
3
4 int MPI_Comm_split(MPI_Comm comm, int color,
5   int key, MPI_Comm *newcomm);
```

Las Figuras 1a-1d-1e muestran un modo de funcionamiento síncrono, en el que los padres detienen la ejecución de la aplicación antes de iniciar la reconfiguración, y la ejecución continua en los hijos al completarse esta. Pero existe un modo de funcionamiento alternativo, denominado asíncrono, en el que la ejecución de los padres continúa mientras la reconfiguración se completa. Las Figuras 1b-1f-1c muestran estas versiones muy similares a las anteriores y donde se aprecia esta particularidad. En estas alternativas, puede ocurrir que los hijos, una vez generados, tengan que esperar hasta que los padres comprueben que ha finalizado la tarea de reconfiguración, así estos pueden acabar la ejecución y los hijos puedan continuarla. En las Figuras 1b-1f-1c, estas esperas están representadas por bloques rayados, mientras que los padres realizan la comprobación en los puntos de control. En estas figuras, también se observa que el modo asíncrono se implementa creando una hebra auxiliar por parte de cada padre, que será la encargada de realizar la creación de los procesos hijos.

B. Comunicaciones MPI para la maleabilidad

En la etapa de redistribución de datos es posible utilizar comunicaciones *P2P* o *COL*. Las primeras se basan en el uso de las operaciones `MPI_Send` y `MPI_Recv` (ver Listado 3), que se pueden utilizar para comunicar cualquier información entre procesos. En este caso, su uso será enviar información de padres a hijos.

Por su parte, las principales operaciones de comunicación colectiva utilizadas para la redistribución de los datos en la maleabilidad son `MPI_Bcast` y `MPI_Alltoallv` (ver Listado 4). Con la operación `MPI_Bcast`, un proceso envía una información al resto de procesos. En la redistribución de datos, esta operación se puede utilizar para enviar el tamaño de los datos o los datos replicados.

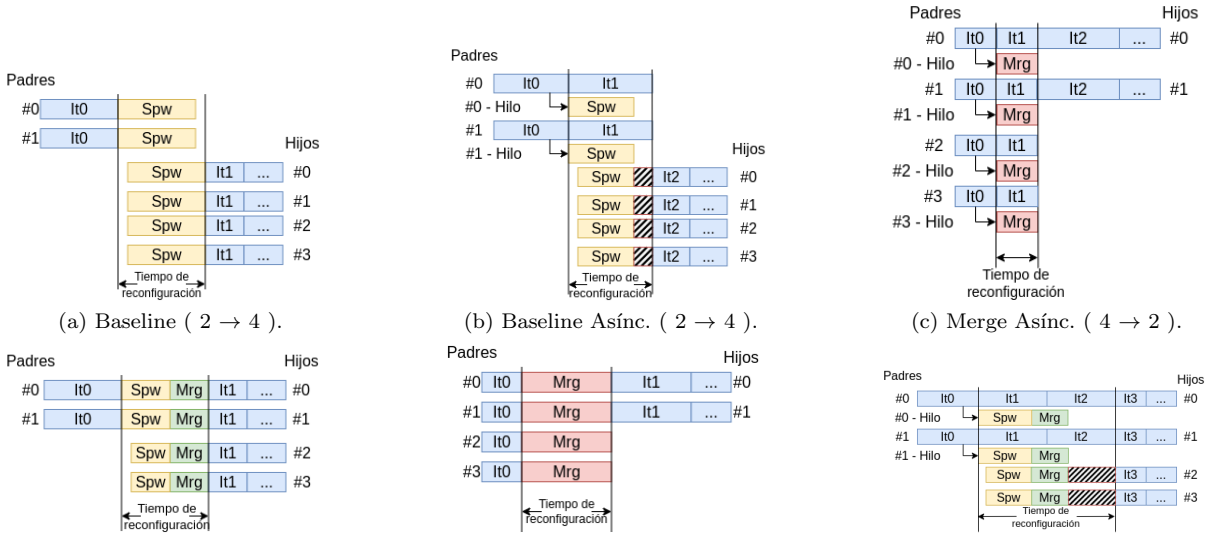
En cambio, la operación `MPI_Alltoallv` es la de mayor coste para la redistribución de datos. Esta función permite que cada proceso envíe/reciba una cantidad de información diferente. Es la operación más utilizada en la etapa 3 de maleabilidad.

Listado 3: Definición de la funciones P2P de MPI para la redistribución de datos. Versiones bloqueantes y no bloqueantes.

```
1 int MPI_Send(const void *buf, int count,
    MPI_Datatype datatype, int dest, int tag,
    MPI_Comm comm);
2 int MPI_Recv(void *buf, int count, MPI_Datatype
    datatype, int source, int tag, MPI_Comm
    comm, MPI_Status *status);
3 int MPI_Isend(const void *buf, int count,
    MPI_Datatype datatype, int dest, int tag,
    MPI_Comm comm, MPI_Request *request);
4 int MPI_Irecv(void *buf, int count, MPI_Datatype
     datatype, int source, int tag, MPI_Comm
    comm, MPI_Request *request)
```

Listado 4: Definición de la funciones COL de MPI para la redistribución de datos. Versiones bloqueantes y no bloqueantes.

```
1 int MPI_Bcast(void *buffer, int count,
    MPI_Datatype datatype, int root, MPI_Comm
    comm);
2 int MPI_Alltoallv(const void *sendbuf, const int
     sendcounts[], const int sdispls[],
    MPI_Datatype sendtype, void *recvbuf, const
     int recvcounts[], const int rdispls[],
    MPI_Datatype recvtype, MPI_Comm comm);
3 int MPI_Ibcast(void *buffer, int count,
    MPI_Datatype datatype, int root, MPI_Comm
    comm, MPI_Request *request);
```



(a) Baseline (2 → 4). (b) Baseline Asínc. (2 → 4). (c) Merge Asínc. (4 → 2).

(d) Merge (2 → 4). (e) Merge (4 → 2). (f) Merge Asínc. (2 → 4).

Fig. 1: Esquema métodos de reconfiguración. Los bloques horizontales son procesos, donde *ItX* es la iteración en ejecución, *Spw* la creación de procesos y *Mrg* fusión de comunicadores. Los bloques rayados indica el tiempo de espera.

```
4 int MPI_Ialltoallv(const void *sendbuf, const
      int sendcounts[], const int sdispls[],
      MPI_Datatype sendtype, void *recvbuf, const
       int recvcounts[], const int rdispls[],
      MPI_Datatype recvtype, MPI_Comm comm,
      MPI_Request *request);
```

El comportamiento de las comunicaciones *P2P* y *COL* varia según el tipo de comunicador que se usa. A continuación se describen dichas diferencias.

B.1 Comunicaciones con intra-comunicadores

Un intra-comunicador se define a partir de un grupo de procesos con un contexto y un nombre de comunicador, como es el caso de *MPI_COMM_WORLD*. Todo proceso del grupo tiene un identificador en el grupo, que se utiliza para definir el origen y el destino de una operación *P2P*.

Existen dos tipos de operaciones *COL*, las que definen un proceso raíz (es decir origen/destino) de la comunicación, y las que consideran que todos los procesos trabajan de igual modo y se especifica que se envía y recibe de cada proceso del comunicador. Entre las primeras destacamos la operación `MPI_Bcast`, donde todos deben indicar al mismo proceso raíz, mientras que `MPI_Alltoallv` es del segundo tipo.

B.2 Comunicaciones con inter-comunicadores

Los inter-comunicadores se componen de dos grupos de procesos con sus contextos y un nombre de comunicador. Las comunicaciones siempre deben realizarse desde uno o más procesos de un grupo hacia uno o más procesos del otro grupo. Es por ello que el origen/destino de una comunicación *P2P* debe ser un identificador de proceso válido en el otro grupo.

Para las operaciones *COL* que definen un proceso raíz, el proceso que envía/recibe la información debe utilizar la constante `MPI_ROOT`, mientras que el resto de procesos de su grupo deben usar `MPI_PROC_NULL`. Por su parte, todos los procesos del otro grupo deben utilizar el identificador del proceso raíz en su grupo.

Para el resto de operaciones *COL*, si los parámetros de la operación definen un patrón de comunicación en el que hay comunicación efectiva en los dos sentidos, esta se debe completar en dos fases. Una primera enviando información del grupo A al B, y a continuación comunicando en sentido inverso.

C. Comunicaciones bloqueantes/no bloqueantes

El estándar define dos tipos de operaciones MPI, *bloqueantes* y *no bloqueantes*, cuyas características pueden ser de interés en la redistribución de datos. A continuación se introducen ambos tipos:

- *Bloqueante*: Aquellas que requieren de una respuesta para poder continuar la ejecución. En el caso de `MPI_Recv`, la respuesta es la recepción del mensaje, pero para `MPI_Send` depende del modo de ejecución. En el modo *synchronous*, la operación finaliza cuando se completa la transferencia, mientras que en el modo *buffered* finaliza tras hacer una copia de la información a enviar. El funcionamiento estándar de `MPI_Send` intenta explotar el modo *buffered*, utilizando los tampones internos de la librería MPI, y si no fuese posible, utiliza el modo *synchronous*.
- *No bloqueante*: Aquellas que no bloquean el flujo de ejecución del programa ya que retornan inmediatamente después de su llamada, sin tener en cuenta si la operación de comunicación se ha completado, por lo que permiten que el proceso realice otras tareas mientras esperan su finalización. El nombre de estas operaciones siguen el formato `MPI_Ixxx`, donde *xxx* representa el nombre de la operación correspondiente.

En los Listados 3 y 4 también se muestran las comunicaciones *no bloqueantes* para *P2P* y *COL*, respectivamente. En estas comunicaciones aparece un argumento adicional respecto a su versión bloqueante, que se asocia con un manejador `MPI_Request` utilizado para consultar el estado de la comunicación. Este manejador se puede usar en la operación `MPI_Wait` para esperar a que finalice la comunicación, o en la operación `MPI_Test`, devolviendo un va-

Tabla I: Algoritmos de MPICH para la función `MPI_Alltoallv`.

Tipo comunicador	Tipo función	
	Bloqueante	**No bloqueante**
Intra-comunicador	Scattered	Schedule Blocked
Inter-comunicador	PairWise Exchange	Schedule PairWise Exchange

lor lógico que indica si la operación se ha completado. Ambas operaciones incluyen una variante con el sufijo *All*, que realizan la espera/validación de varias comunicaciones no bloqueantes.

D. Algoritmos de comunicación colectiva para Alltoallv en MPICH

Buena parte de los datos en las aplicaciones paralelas se encuentran distribuidos entre todos los procesos, por lo que para su redistribución se requerirá el uso de la operación `MPI_Alltoallv`. A continuación se realiza un análisis más detallado de como se lleva a cabo esta operación en el estándar de MPICH [9].

La Tabla I clasifica los diferentes algoritmos utilizados para implementar `MPI_Alltoallv`, según el tipo de comunicador utilizado y si la operación se usa en modo *bloqueante* o *no bloqueante*. A continuación se describen sus principales características:

- *Scattered*: Algoritmo bloqueante para intra-comunicadores basado en `Irecv` + `Isend`. El total de comunicaciones de un proceso se divide en conjuntos. Para cada conjunto se hacen hasta 32[1] comunicaciones y una llamada a `MPI_Waitall`. Los procesos solo realizan comunicaciones con aquellos a los que tienen que comunicar datos. En el Algoritmo 1 se muestra una simplificación del mismo.
- *Schedule Blocked*: Variante no bloqueante del algoritmo *Scattered* para intra-comunicadores. El algoritmo es el mismo pero no se utiliza `MPI_Waitall` para que el proceso pueda continuar con la realización de otras tareas. Por tanto, la llamada termina para un proceso cuando crea sus solicitudes de comunicación.
- *PairWise Exchange*: Algoritmo bloqueante para inter-comunicadores basado en `MPI_Sendrecv`. Cada proceso del grupo A realiza tantas comunicaciones como procesos haya en el grupo B y viceversa (ver Algoritmo 2). El algoritmo no comprueba si hay datos que enviar a ese proceso antes de realizar la llamada a `MPI_Sendrecv`.
- *Schedule PairWise Exchange*: Versión no bloqueante del algoritmo *Schedule Blocked* pero para inter-comunicadores. La única diferencia es que no se hace por conjuntos ni tampoco comprueba si hay bytes a comunicar antes de realizar una solicitud de comunicación.

III. Redistribución de datos

En esta sección, se describen los diferentes algoritmos y estrategias utilizados para redistribuir la información desde los procesos padre hacia los hijos para que estos continúen con la ejecución (Etapa 3 de la maleabilidad).

[1]Por defecto el valor por conjunto en el algoritmo *Scattered* es de 32, pero es posible modificarlo.

Algoritmo 1 Simplificación del algoritmo *Scattered*.

```
// Validaciones previas
b = 32 // Valor por defecto del bloque
c_size = MPI_Comm_size
for ( ii = 0; ii < c_size; ii+=b ) do
    ss = c_size − ii < b ? c_size − ii : b
    for ( i = 0; i < ss; i++ ) do
        dst = (myId + i + ii) %c_size;
        MPI_Irecv: dst → myId
        dst = (myId − i − ii + c_size) %c_size;
        MPI_Isend: myId → dst
    end for
    MPI_Waitall // Solo en algoritmo bloqueante
end for
```

Algoritmo 2 Simplificación del algoritmo *PairWise Exchange*.

```
// Validaciones previas
l_size = MPI_Comm_size
r_size = MPI_Comm_remote_size
m_size = max: r_size, l_size
for ( i = 0; i < m_size; i++ ) do
    dst = (myId + i) %m_size
    src = (myId − i + m_size) %m_size
    if ( dst ≥ r_size ) then dst = MPI_PROC_NULL
    end if
    if ( src ≥ r_size ) then src = MPI_PROC_NULL
    end if
    MPI_Sendrecv: myId → dst, src → myId
end for
```

Esta sección esta estructurada en dos partes. En primer lugar, se analizan algunas cuestiones básicas relacionadas con la redistribución de datos. A continuación, se presentan los distintos métodos implementados para completar esta etapa. El conjunto de métodos se basan en el tipo de comunicación, síncrona o asíncrona que a su vez utilizarán operaciones del tipo *P2P* o *COL*.

A. Conceptos previos

La redistribución de los datos se realiza junto con el redimensionado del número de procesos. Primero se generan/liberan los procesos y después se realiza la redistribución. Es por ello que en las imágenes que aparecen en la Figura 1, la redistribución debería aparecer a continuación de la Etapa 2 y siendo una parte del tiempo de reconfiguración de la aplicación.

En la redistribución de datos hay varios aspectos a considerar. Uno de ellos es la estructura interna de la información a comunicar, que suelen ser vectores, matrices densas y matrices dispersas. Otro es el modo en el que esta información aparece distribuida entre los procesos. El uso de un reparto estático por bloques facilita esta labor en los dos primeros casos, pero en nada ayuda en el último caso, ya que el número de elementos no nulos que se debe comunicar entre dos procesos puede variar mucho. Esta situación es todavía más acusada en repartos dinámicos y/o con estructuras de datos más complejas.

Es por ello, que generalizar el patrón de comunicación entre procesos no es una tarea fácil. En muchos casos, la comunicación se divide en dos fases: en la primera, cada padre envía a los hijos el tamaño de la información a comunicar, para que estos puedan crear las estructuras necesarias, mientras que en la segunda fase, se realiza la comunicación de la infor-

Algoritmo 3 Estructura básica de redistribución de datos.

```
// Padres envían tamaños a hijos
if ( myId ≥ frstChd && myId ≤ lstChld ) then
    Los hijos crean estructuras internas
end if
// Padres envían información a hijos
```

Algoritmo 4 Inicialización vectores de comunicación.

```
r_size = Malleability_Other_group
counts = calloc: → r_size
displs = calloc: → r_size
for ( i = 0; i < r_rsize; i++ ) do
    r_ini, r_end = Block_id: → i
    if (ini ≥ r_end||end ≤ r_ini) then continue
    end if
    big_ini = ini > r_ini?ini : r_ini
    small_end = end < r_end?end : r_end
    counts[i] = small_end − big_ini
    displs[i + 1] = displs[i] + counts[i]
end for
```

mación completa. El Algoritmo 3 muestra esquemáticamente la estructura básica de la redistribución.

B. Cálculos relacionados con la redistribución

Como tarea previa a la comunicación de una estructura distribuida, cada padre/hijo debe calcular dos vectores en el que se indique cuantos elementos va a enviar/recibir a/de cada padre/hijo. Aquellos procesos que tengan ambos roles deberán realizar este calculo dos veces, uno por cada rol.

Los vectores a calcular son *cuentas* (*counts*) y *desplazamientos* (*displs*). El tamaño de los vectores para los padres es igual al número de hijos, y viceversa para los hijos. El primer vector indica el número de elementos a comunicar a/desde cada proceso destino/origen. Por su parte, el segundo vector marca la posición del primer elemento a comunicar al citado proceso.

En el Algoritmo 4 se muestra como calcular ambos vectores para un proceso padre/hijo. Asumiendo una distribución por bloques, los elementos en un proceso son los comprendidos en el intervalo $[ini, end]$. Para calcular los elementos a comunicar, cada proceso padre/hijo debe calcular la intersección con la de los correspondientes hijos/padres, que indican el número de elementos a comunicar a/de cada proceso. La función `Malleability_Other_group` obtiene el número de procesos padres/hijos, mientras que la función `Block_id` obtiene los valores *ini* y *end* de un proceso remoto, padre/hijo.

C. Métodos síncronos

Estos métodos se pueden implementar utilizando operaciones *P2P* o *COL*.

Los métodos síncronos basados en *P2P* utilizan las funciones `MPI_Send` y `MPI_Recv`. Aunque ambas sean operaciones bloqueantes, su uso junto al método *Baseline* imposibilita la aparición de cualquier tipo de interbloqueo, ya que la intersección entre padres e hijos es vacía. Pero el uso de estas funciones junto al método *Merge* puede ser más problemático, ya que la intersección no es vacía; aún así la utilización del modo *buffered* de `MPI_Send` resolvería muchos de los posibles interbloqueos. En cualquier caso, la versión más segura combinaría el uso de la

Algoritmo 5 Redistribución utilizando operaciones *P2P*.

```
if (myId ≥ frsPrnt && myId ≤ lstPrnt) then
    // Padres envían información
    for ( i = frstChd; i ≤ lstChld; i++ ) do
        if ( i == myId ) then
            Copia local usando memcpy
        else
            MPI_Isend: myId → i, tag = 77
        end if
    end for
end if
if ( myId ≥ frstChd && myId ≤ lstChld ) then
    // Hijos reciben información
    for ( i = frsPrnt; i ≤ lstPrnt; i++ ) do
        if ( i ≠ myId ) then
            MPI_Recv: MPI_ANY_SOURCE , tag = 77
        end if
    end for
end if
if (myId ≥ frsPrnt && myId ≤ lstPrnt) then
    // Padres esperan la finalización
    MPI_Waitall: tag = 77
end if
```

Algoritmo 6 Redistribución utilizando operaciones *COL*.

```
// Procesos envían/reciben información
MPI_Alltoallv: parents → children
```

función `MPI_Isend` con `MPI_Waitall`/`MPI_Testall`. El Algoritmo 5 muestra como podría implementarse esta comunicación, donde los identificadores de los procesos padre e hijo se definen, respectivamente, en dos intervalos $[frsPrnt, lstPrnt]$ y $[frsChld, lstChld]$, respectivamente. La utilización adecuada de `MPI_ANY_SOURCE` acelera la recepción de los mensajes por parte de los hijos. Además, el uso de valores diferentes de etiqueta (`tag`) permitiría realizar consecutivamente varias comunicaciones con un único control de finalización al final del algoritmo.

Los métodos síncronos basados en operaciones *COL*, por su propio diseño, no provocan interbloqueos. Este se basa en la utilización de las funciones `MPI_Alltoall` o `MPI_Alltoallv` en su modo *bloqueante*. Esta alternativa simplifica el código necesario para la redistribución de datos desde los procesos padre a los hijos, puesto que no hay que comprobar la intersección entre padres e hijos y no hay que hacer copias locales. El Algoritmo 6 muestra esta implementación.

D. Métodos asíncronos

Para este tipo de comunicación, una opción es utilizar *hebras auxiliares* que son las responsables de llevar a cabo la redistribución de datos, liberando a la hebra principal de esta tarea. Estas hebras utilizarán alguna de las opciones descritas con anterioridad en el Algoritmos 5 o 6. Otra alternativa es el uso de las versiones *no bloqueantes* de las operaciones de comunicación. En el caso del Algoritmo 5, la solución es separar la fase de comunicación de la fase de finalización, que se debería realizar en los puntos de sincronización elegidos por la aplicación. Además habría que sustituir `MPI_Recv` por `MPI_Irecv` y `MPI_Waitall` por `MPI_Testall`. Hay que tener en cuenta, que el primer cambio es obligatorio cuando los procesos han sido generados con el método *Merge*, ya que un padre puede participar como hijo al mismo tiempo.

Por su parte, en el Algoritmo 6 se debería utilizar `MPI_Ialltoallv` para iniciar la comunicación, y `MPI_Testall` en cada punto de sincronización para comprobar la finalización de las mismas.

IV. Resultados experimentales

En esta sección se presentan los experimentos y análisis realizados para comparar los métodos descritos en la Sección III. Las conclusiones de este estudio serán fundamentales para implementar una versión optimizada de las etapas 2 y 3 de la reconfiguración de una aplicación maleable.

A. Hardware y Software utilizados

Los experimentos se han realizado en un clúster de ocho nodos con dos procesadores Intel Xeon 4210 de 10 núcleos cada uno, por tanto con un total de 160 núcleos. Los nodos están interconectados con una red Infiniband EDR de 100GB/s y con una Ethernet de 10GB/s, utilizándose una versión diferente de MPI para trabajar con cada red. Por un lado, se utiliza la versión MPICH 4.0.3 [9] compilada con CH4:OFI netmod (Infiniband) y, por otro, la versión MPICH 3.4.1 se ha compilado con CH3:Nemesis netmod (Ethernet).

En el estudio se han realizado 10 ejecuciones de una aplicación que solo realiza dos tareas: creación de procesos y redistribución de datos. En la primera tarea se utilizan los métodos descritos en el apartado II-A, mientras que en la segunda tarea se redistribuyen 5Gb de información desde NP padres a NH hijos, utilizando las diferentes estrategias de redistribución de datos descritas en la Sección III. El estudio consta de una reconfiguración por experimento, que se inician con 2, 20, 40, 80, 120 y 160 procesos padre y se reconfiguran a los mismos números de procesos hijo, dando lugar a 30 (NP, NH) pares. El número de nodos del clúster ocupados en cada ejecución se calcula como $\lceil N/20 \rceil$, donde N es el máximo entre NP y NH, lo que permite minimizar los recursos asignados por el RMS.

Para completar los experimentos, se han utilizado las operaciones *P2P* y *COL* en sus variantes *síncrona* (comunicación bloqueante) y *asíncrona* (comunicación no bloqueante). En este último caso, aunque la comunicación sea del tipo asíncrono, no se solapa con ningún otro cómputo, porque el objetivo es analizar el comportamiento de la comunicación entre padres e hijos, y descubrir qué alternativa es la más eficiente cuando únicamente se realiza la redistribución. Por su parte, el método de creación de procesos determina el tipo de comunicador a utilizar. Por defecto, *Baseline* utiliza inter-comunicadores y *Merge* maneja intra-comunicadores, aunque también se ha implementado una variante de *Baseline* que utiliza intra-comunicadores, *BaselineIntra*. Finalmente, el análisis se ha realizado para los dos tipos de redes que dispone el sistema, Ethernet e Infiniband.

B. Tiempos de redistribución

En este apartado se realiza un estudio sobre el tiempo de redistribución de $5Gb$ de datos en las dos redes objeto de análisis, Infiniband y Ethernet, considerando diferentes alternativas de comunicación y operación. Estas alternativas se han obtenido al combinar las diferentes estrategias implementadas para llevar a cabo las etapas 2 y 3 de la maleabilidad con distintos valores de pares (NP, NH).

Las Figuras 2 y 3 muestran el tiempo de redistribución sobre Ethernet e Infiniband, respectivamente. En ambas figuras, se han separado los pares asociados al aumento del número de procesos (expansión), que aparecen en la gráfica superior, y los pares asociados a la reducción del número de procesos (reducción), que aparecen en la gráfica inferior. Además, el eje X muestra los diferentes pares (NP, NH) considerados, mientras que el eje Y indica el coste de la comunicación en segundos. Por su parte, las leyendas indican el método de reconfiguración utilizado, (*Baseline*, *BaselineIntra* y *Merge*), el tipo de operación (*P2P* o *COL*) y si se ha utilizado una variante *Síncrona* o *Asíncrona* (marcado por el sufijo S o A, respectivamente).

Al analizar las figuras asociadas a la **expansión**, se observa que todas las variantes tienen un comportamiento muy similar independientemente del número de procesos, excepto en la variante *Baseline-COLS*, cuya diferencia de tiempos es grande y oscila entre $1s$ y $8,5s$. Este comportamiento se justifica por la implementación de la operación `MPI_Alltoallv` sobre inter-comunicadores en el compilador MPICH, que utiliza el algoritmo `PairWise Exchange`, basado en comunicación de tipo S, en el que se requiere que cada proceso termine una comunicación para realizar la siguiente. Por su parte, el resto de algoritmos *P2P* o *COL* están basados, parcial o completamente, en comunicaciones *no bloqueantes* que pueden solapar comunicaciones.

También resulta destacable una ligera mejora entre $0,2s$ y $0,3s$ en las variantes *Merge* sobre las *Baseline* en la red Infiniband, independientemente del tipo de comunicador, debido, principalmente, a la aparición de la suscripción en exceso en estos últimos. En los experimentos sobre la red Ethernet solo se observa este efecto para valores grandes de NP (80 ó 120).

El comportamiento de la **reducción** cuando $NH \geq 20$ es similar al descrito en la expansión, observándose que la variante *Baseline-COLS* necesita un tiempo mayor que el resto de variantes, con una diferencia que oscila desde $3s$ a $8s$, justificado por el uso del algoritmo *PairWise Exchange* en la implementación de la operación. Además, las variantes *Merge* en ambas redes, obtienen tiempos ligeramente mejores por una diferencia de $0,1s$ y desde $0,1s$ a $0,4s$ en Ethernet e Infiniband, respectivamente, debido a la aparición de la suscripción en exceso en las variantes *Baseline*.

Sin embargo, el comportamiento cuando $NH = 2$ es irregular en todas las variantes, siendo las más rápidas en la red Ethernet *Merge-COLA* y *Merge-*

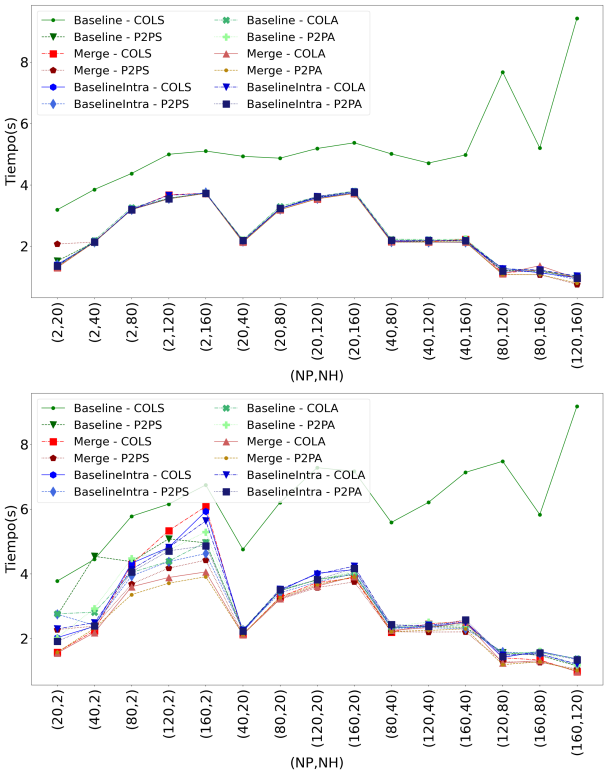


Fig. 2: Tiempo de redistribución de 5Gb de datos en Ethernet desde NP a NH. Expansión (arriba), Reducción (abajo).

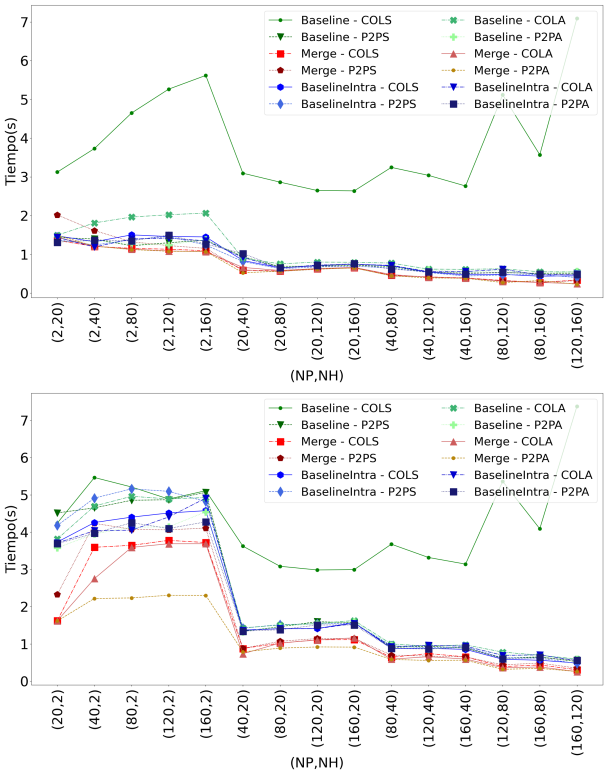


Fig. 3: Tiempo de redistribución de 5Gb de datos en Infiniband desde NP a NH. Expansión (arriba), Reducción (abajo).

P2PA, con una diferencia máxima de $0,4s$ respecto a la variante *Merge-P2PS*. En cambio, la variante más rápida en la red Infiniband es *Merge-P2PA*, con una diferencia máxima de $1,4s$ respecto a la variante *Merge-COLA*. Este comportamiento se puede justificar por la congestión de mensajes en los hijos, que deben recibir mensajes desde muchos padres.

La comparación de las variantes *Baseline* y *BaselineIntra* permite analizar el impacto del uso de **intra-** e **inter-comunicadores**. En este análisis, solo se detectaron diferencias significativas en la variante *Baseline-COLS*, ya que es la única que utiliza el algoritmo *PairWise Exchange*, basado en comunicaciones bloqueantes, que no genera solapamiento de comunicaciones. El resto de variantes utilizan algoritmos basados en comunicaciones asíncronas: *Scattered* y *Schedule Blocked* para intra-comunicadores, mientras que *Schedule PairWise Exchange* utiliza la variante *no bloqueante* sobre inter-comunicadores.

C. Sobrecoste de variantes asíncronas

En este apartado se realiza un estudio del sobrecoste que tienen las variantes asíncronas (A) respecto a las variantes síncronas (S). Este sobrecoste, que se refleja en el valor de α, se ha calculado como el cociente de cada una de las variantes A con su respectiva variante S. Así, valores superiores a 1 indican que la variante A es más costosa que la variante S asociada, y valores menores que 1 es justo lo contrario.

Las Figuras 4 y 5 muestran los valores de α con la red Ethernet e Infiniband, respectivamente. La parte superior de ambas figuras es para el caso de expansión de procesos, y la parte inferior para reducción. La interpretación de las leyendas y los valores del eje X también son idénticas al descrito en el estudio anterior. En este caso, únicamente varía el significado del eje Y que indica el valor de α.

En el caso de la **expansión** para la red Ethernet, los valores de α siempre tienen una diferencia menor a $\pm 5\,\%$ respecto al valor 1. Solo en la variante *Baseline-COLA* existe una reducción del tiempo de redistribución entre el $30\,\%$ y $70\,\%$ debido al mayor coste de la variante *Baseline-COLS*.

Se observa un comportamiento más irregular para la red Infiniband, donde la diferencia llega hasta $\pm 10\,\%$ respecto al valor 1, siendo en 4 de 75 casos mayor a $\pm 10\,\%$, exceptuando la variante *Baseline-COLA*. Pero a partir de 20 procesos padre los valores se estabilizan alrededor de 1.

El análisis de la **reducción** muestra dos situaciones diferentes. La primera se produce cuando ($NH = 2$), donde aparece un comportamiento irregular de todas las variantes para ambas redes, debido a la congestión de mensajes ya mencionada con anterioridad.

Por su parte, la segunda situación se observa cuando ($NH > 2$) y es dependiente de la red. En la red Ethernet, los valores de α siempre tienen una diferencia menor a $\pm 10\,\%$ respecto al valor 1. En cambio, en 11 de las 20 combinaciones posibles de las variantes *Merge* asíncronas sobre la red Infiniband, el valor de α tiene una diferencia mayor a $\pm 10\,\%$ respecto a 1. Además, 9 de estos 11 casos son de la variante *Merge-P2PA*, que tiene un menor coste, ya que utiliza comunicaciones no bloqueantes.

Finalmente, comentar que la variante *Baseline-COL* siempre obtiene valores de α menores a 1, independientemente de la red. Por tanto, resulta aconsejable utilizar *Baseline-COLA* antes que *Baseline-*

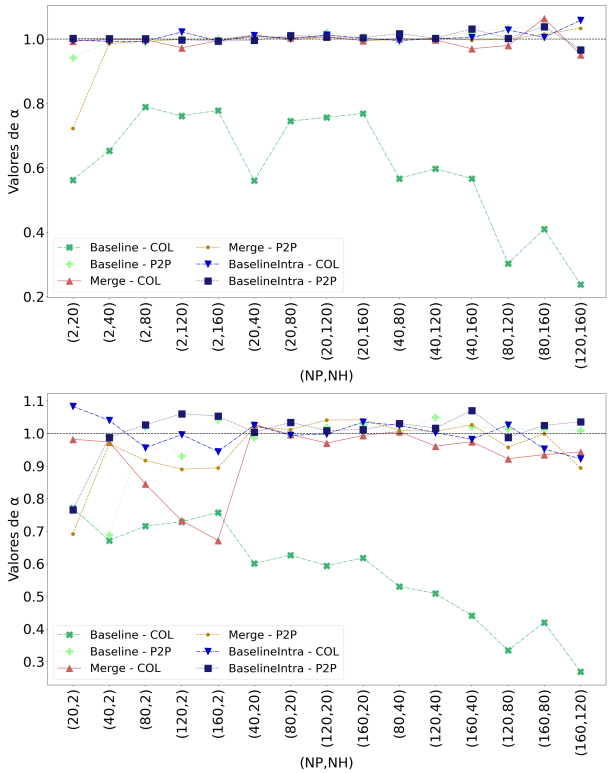


Fig. 4: Valores de α para la redistribución de $5Gb$ de NP a NH en Ethernet. Expansión (arriba), Reducción (abajo).

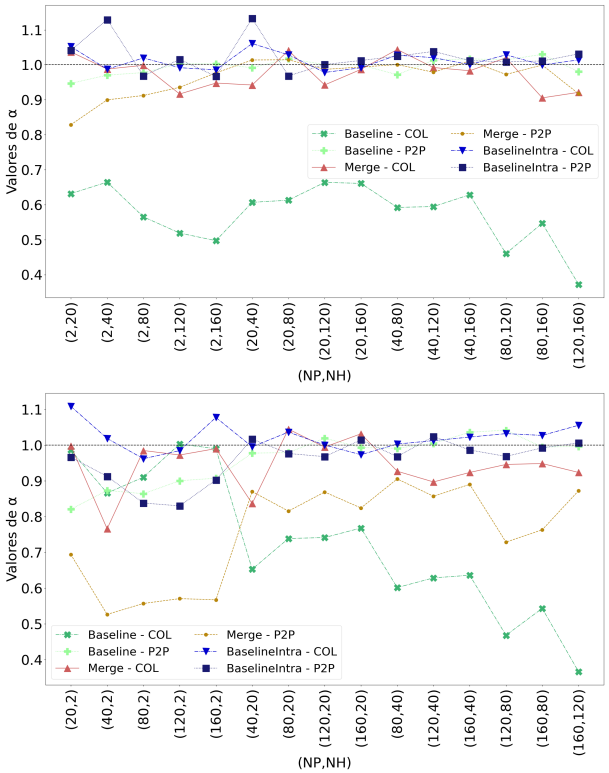


Fig. 5: Valores de α para la redistribución de $5Gb$ de NP a NH en Infiniband. Expansión (arriba), Reducción (abajo).

COLS, aún cuando se produzca en el primero cierto retraso en el inicio de la ejecución de los hijos, puesto que los padres tienen que alcanzar el punto de control y comprobar el final de la comunicación.

D. Análisis de las variantes de redistribución

En este apartado se ha realizado un estudio estadístico para determinar que variante es la más rápida para realizar una redistribución de datos desde NP padres a NH hijos.

Para este estudio se han utilizado las pruebas de Shapiro-Wilk [10], Kruskal-Wallis [11] y Post hoc Connover-Iman [12]. Al realizar la primera se concluye que todas las variantes rechazan la hipótesis nula (H0), en la que cada variante provendría de una distribución normal, y por tanto son necesarias pruebas no paramétricas para comparar las variantes.

Con la prueba de Kruskal-Wallis se comprueba para cada par (NP, NH), si todos las variantes provienen de la misma población (es decir, tienen la misma mediana, H0), o al menos una de ellas proviene de una distribución diferente (H1). Para los casos en los que se rechaza H0 se realiza la prueba de Post hoc Connover-Iman para descubrir qué variantes son diferentes para ese mismo par (NP, NH).

La Figura 6 muestra en una cuadrícula la variante óptima para realizar una distribución de datos para cada par (NP, NH), tanto para Ethernet (Izquierda) como para Infiniband (Derecha). El triangulo superior se asocia con las expansiones de la aplicación, mientras que el triangulo inferior lo hace con las reducciones. El número y color de cada celda muestran la variante más rápida para realizar la redistribución según las pruebas estadísticas. En caso de igualdad en una celda, se selecciona aquella variante que haya sido más seleccionada en el resto de celdas.

Una primera observación de la figura permite concluir que la variante *Merge-P2PA* es la más rápida en cualquiera de las dos redes, excepto tres excepciones en el caso de Ethernet y una para Infiniband.

La elección de las variantes *Merge*, respecto de las variantes *Baseline* y *BaselineIntra*, se justifica por la suscripción en exceso que generan estos últimos y que hace que aumente su tiempos de operación.

Por su parte, la elección de las comunicaciones *P2P* en lugar de las *COL*, se justifica por una diferencia significativa en sus medianas. La implementación de la variante *Merge-P2P* es muy similar al algoritmo *Scattered* de la función `MPI_Alltoallv` de MPICH utilizado en las variantes *Merge-COL*. Es por ello que las validaciones que se realizan al inicio del algoritmo *Scattered* son la principal razón que puede justificar esta diferencia.

Las tres excepciones de la red Ethernet se producen al reducir de $NP = 80$ o más procesos padre a $NH = 20$. En estos casos la variante más eficiente es la *Merge-P2PS*. En cambio, la excepción de la red Infiniband aparece al expandir de $NP = 2$ a $NH = 20$ procesos, siendo la variante más rápida la *Merge-COLS*. Estas excepciones ocurren porque el tiempo máximo de la variante *Merge-P2PA* ha sido superior al de las excepciones con subidas desde $0,2s$ a $0,6s$, pero en todos los casos la cota inferior de *Merge-P2PA* es mejor por $0,1s$.

V. Conclusiones

En este artículo se han analizado 12 variantes diferentes para realizar la etapa de redistribución de datos durante la reconfiguración de una aplicación MPI maleable. Primero se presentan 2 variantes, una ba-

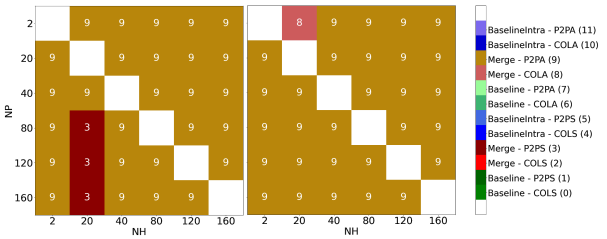


Fig. 6: Variantes óptimas para redistribuir datos en Ethernet (Izquierda) e Infiniband (Derecha).

sada en comunicaciones punto a punto (*P2P*) y otra en comunicaciones colectivas (*COL*). A continuación se indica como utilizar dichas variantes utilizando la versión *síncrona* o *asíncrona* de estas operaciones, creando dos alternativas adicionales.

Además, las variantes pueden tener comportamientos diferentes según el método utilizado en la creación de procesos: *Baseline* o *Merge*.Pero es posible seleccionar si el método *Baseline* usa intercomunicadores o intra-comunicadores, mientras que con el método *Merge* solo es posible utilizar intracomunicadores, dando lugar a tres nuevas alternativas.

También hay que destacar que los métodos de redistribución basados en comunicaciones *COL* solo requieren realizar llamadas a funciones colectivas de MPI, mientras que para las comunicaciones *P2P*, se han tenido que diseñar dos algoritmos basados en operaciones no bloqueantes para evitar interbloqueos.

El estudio experimental de las variantes realiza la redistribución de $5Gb$ de datos desde los procesos padre a los hijos, en una aplicación que solo lleva a cabo las etapas 2 y 3 de la maleabilidad, analizando el comportamiento en dos redes existentes en el sistema, Ethernet 10G e Infiniband EDR. El análisis de los tiempos permite concluir que la variante *Baseline-COLS* presenta un coste más elevado que el resto, debido al algoritmo que utiliza MPICH para implementar `MPI_Alltoallv` sobre inter-comunicadores que se basa en operaciones bloqueantes. El resto de variantes convergen a tiempos similares con diferencias máximas de $0,6s$, salvo los casos de reducción de procesos con ($NH = 2$), en los que aparece un comportamiento irregular debido a la congestión que sufren los dos procesos hijo.

Finalmente se han comparado estadísticamente todos las variantes modificando los valores de NP y NH, obteniendo que la variante *Merge-P2PA* es la óptima en 56 de 60 casos. Hay tres razones que justifican esta elección: (i) evitar el estado de suscripción en exceso del método *Baseline*; (ii) realizar las comunicaciones sin un orden predeterminado, gracias al uso de comunicaciones no bloqueantes; y (iii) evitar las validaciones internas de la operación `MPI_Ialltoallv`, que no se realizan al utilizar operaciones *P2P*.

El trabajo futuro se centrará en analizar el efecto de solapar la realización de las etapas 2 y 3 de la maleabilidad con el procesamiento de una aplicación científica, y su impacto sobre las prestaciones. También se diseñarán nuevos métodos de redistribución de datos basadas en comunicaciones RMA (*Remote Memory Access*) de MPI.

Agradecimientos

El presente trabajo ha sido subvencionado por el proyecto PID2020-113656RB-C21 financiado por MCIN/AEI/10.13039/501100011033. El trabajo del investigador I. Martín-Álvarez fue subvencionado por la ayuda predoctoral ACIF/2021/260, financiada por el Gobierno Autonómico Valenciano y por la European Social Funds.

Referencias

[1] Message Passing Interface Forum, *MPI: A Message-Passing Interface Standard Version 4.0*, June 2021.

[2] Jonas Posner and Claudia Fohry, "Transparent resource elasticity for task-based cluster environments with work stealing; transparent resource elasticity for task-based cluster environments with work stealing," *50th International Conference on Parallel Processing Workshop*, 2021.

[3] Sergio Iserte, Héctor Martínez, Sergio Barrachina, Maribel Castillo, Rafael Mayo, and Antonio J Peña, "Dynamic Reconfiguration of Noniterative Scientific Applications," *The International Journal of High Performance Computing Applications*, p. 109434201880234, sep 2018.

[4] Sergio Iserte, Rafael Mayo, Enrique S. Quintana-Ortí, Vicenç Beltran, and Antonio J. Peña, "DMR API: Improving Cluster Productivity by Turning Applications into Malleable," *Parallel Computing*, vol. 78, pp. 54–66, oct 2018.

[5] Gonzalo Martín, David E. Singh, Maria-Cristina Marinescu, and Jesús Carretero, "Enhancing the Performance of Malleable MPI Applications by Using Performance-aware Dynamic Reconfiguration," *Parallel Computing*, vol. 46, pp. 60–77, jul 2015.

[6] Chao Huang, Orion Lawlor, and L. V. Kalé, "Adaptive MPI," in *Proceedings of the 16th International Workshop on Languages and Compilers for Parallel Computing (LCPC 2003), LNCS 2958*, College Station, Texas, October 2003, pp. 306–322.

[7] Iker Martín-Álvarez, José I Aliaga, Maribel Castillo, Sergio Iserte, and Rafael Mayo, "Dynamic spawning of mpi processes applied to malleability," *The International Journal of High Performance Computing Applications*, vol. 0, no. 0, pp. 10943420231176527, 0.

[8] Nicholas Radcliffe, Layne Watson, and Masha Sosonkina, "A comparison of alternatives for communicating with spawned processes," in *Proceedings of the 49th Annual Southeast Regional Conference*, New York, NY, USA, 2011, ACM-SE '11, p. 132–137, Association for Computing Machinery.

[9] MPICH Development team, "Mpich website," `https://www.mpich.org/`.

[10] S. S. Shapiro and M. B. Wilk, "An analysis of variance test for normality (complete samples)," *Biometrika*, vol. 52, no. 3/4, pp. 591–611, 1965.

[11] W.H. Kruskal and W.A. Wallis, "Use of ranks in one-criterion variance analysis," *Journal of the American Statistical Association*, vol. 47, no. 260, pp. 583–621, 1952.

[12] W J Conover and R L Iman, "Multiple-comparisons procedures. informal report," 2 1979.