DEPARTAMENTO
DE COMPUTACION
Facultad de Ciencias Exactas y Naturales - UBA

# Trabajo Práctico III

**Grupo: 6**

Sistemas Operativos
Primer Cuatrimestre 2017


| Integrante | LU | Correo electrónico |
|---|---|---|
| Emiliano Galimberti | 109/15 | emigali@hotmail.com |
| Ignacio M. Fernandez | 47/14 | nachofernandez.1995@hotmail.com |
| Pedro Mattenet Riva | 428/15 | peter_matt@hotmail.com |






Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires
Ciudad Universitaria - (Pabellón I/Planta Baja)
Intendente Güiraldes 2160 - C1428EGA
Ciudad Autónoma de Buenos Aires - Rep. Argentina
Tel/Fax: (54 11) 4576-3359
http://www.fcen.uba.ar

# Índice


**1. Introducción** **3**
1.1. Función nodo . . . . . 3

**2. Ejercicio1** **4**
2.1. load . . . . . 4
2.1.1. Consola . . . . . 4
2.1.2. Nodo . . . . . 4

**3. Ejercicio 2** **5**
3.1. addAndInc . . . . . 5
3.1.1. Consola . . . . . 5
3.1.2. Nodo . . . . . 5

**4. Ejercicio 3** **6**
4.1. member . . . . . 6
4.1.1. Consola . . . . . 6
4.1.2. Nodo . . . . . 6

**5. Ejercicio 4** **6**
5.1. maximum . . . . . 6
5.1.1. Consola . . . . . 7
5.1.2. Nodo . . . . . 7

**6. Ejercicio 5** **7**
6.1. quit . . . . . 7
6.1.1. Consola . . . . . 8
6.1.2. Nodo . . . . . 8

**7. Experimentación** **8**

# 1. Introducción

En este trabajo practico se pidió implementar las funciones del `ConcurrentHashMap` utilizando una interfaz de MPI para profundizar el tema de sistemas distribuidos. La estructura del informe presenta una sección para cada ejercicio y al final se presenta una experimentación realizada con los archivos `corpus` provistos por la cátedra y además otros archivos que creamos nosotros para experimentar.

El parámetro `np` indica la cantidad de nodos disponibles para realizar el trabajo de forma distribuida. Cada nodo tiene un rank. El nodo con rank 0 es la consola y el resto de los ranks ( de 1 a np) son los nodos que se utilizan para ejecutar las funciones del HashMap. Si se utiliza $\texttt{np} < 2$ no se pueden ejecutar las funciones pues no hay nodos disponibles.

Para distinguir las funciones cuando la consola le envía mensajes a los nodos utilizamos el tag. Creamos una variables llamada `funciones` para distinguir los trabajos que los nodos deben realizar. Los tags son los siguientes :

- **tag 1 :** representa la función `load`
- **tag 2 :** representa la función `addAndInc`
- **tag 3 :** representa la función `member`
- **tag 4 :** representa la función `maximum`
- **tag 5 :** representa la función `quit`
- **tag 90:** lo utilizan los nodos para comunicarle a la consola que terminaron de enviar todo su contenido (maximum).
- **tag 99 :** lo utilizan los nodos para comunicarse con la consola.

## 1.1. Función nodo

`nodo` es una función que recibe como parámetro su rank. La estructura general de esta función es la siguiente :

1. Se crea un Hash Map localmente.
2. Se crean los buffers `buf` y `bufi` para mandarse mensajes con la consola.
3. Se genera un ciclo con la guarda `while(true)` donde el nodo espera un mensaje de la consola. Esto se hace mediante la función `MPI_Probe` y la estructura `status`. Esta función nos permite capturar la longitud del mensaje que envió la consola junto a su tag. Luego en base a este ultimo parámetro se realizara la función correspondiente al tag. Cada una estará explicada en las secciones posteriores.

# 2. Ejercicio1

## 2.1. load

El ejercicio consiste en pasar por consola una lista de archivos. Estos archivos poseen muchas palabras que se agregaran a nuestro HashMap. Se pide utilizar un nodo para cada archivo. Y en caso de haber mas archivos que nodos, se debera enviar un archivo a cada nodo a medida que vaya liberándose.

### 2.1.1. Consola

Para un mejor control sobre los nodos, utilizamos como estructura auxiliar una cola de enteros ($queue < int >$) llamada nodosIdle. Esta cola va a representar los nodos que estan sin tareas.

Lo primero que hacemos es inicializar la cola insertando todos los nodos. Luego iteramos sobre la lista de archivos que recibimos como parámetro. Para cada caso, se decide:

- **Si la cola de nodos disponibles NO esta vacía:** Se desencola el nodo a trabajar. Se asigna un buffer de memoria con el nombre del archivo de la iteración, y se envía mediante *MPI_Send* (bloqueante).
- **Si la cola de nodos disponibles está vacía** Entonces esperamos a que alguno de los nodos termine de trabajar. Como no sabemos que nodo va a terminar primero, utilizamos *MPI_Probe* para saber el origen del próximo mensaje que recibamos (de que un nodo termino). Una vez recibido el mensaje y ubicado el nuevo nodo disponible, lo insertamos en la cola y seguimos en el ítem anterior.

Una vez finalizado el ciclo, vamos a ciclar en $while$ hasta que nodosIdle este completo, o sea, el tamaño de la cola sea igual a la cantidad de nodos.

Dentro del $while$ vamos a hacer el mismo mecanismo que en el caso de nodosIdle vació. Con *MPI_Probe* detectamos el próximo nodo a terminar, lo recibimos y lo agregamos a la cola.

Terminamos con la lista de archivos distribuida por los nodos y todos los nodos disponibles nuevamente.

### 2.1.2. Nodo

Una vez que sabemos que el próximo mensaje es una tarea de LOAD, lo recibimos con *MPI_Recv*, y almacenamos el nombre del documento a guardar en el buffer previamente pedido.

Ahora aplicamos a nuestro HashMap la funcion $load()$, pasando como parámetro lo que guardamos en nuestro buffer. Trabajamos arduamente y liberamos el buffer. Ahora resta comunicarle a la consola que terminamos nuestro trabajo. Para eso, pedimos otro buffer de tamaño de un entero, para poder guardar nuestro numero de nodo en el y mandárselo a la consola. Enviamos el mensaje por medio de *MPI_Send* y liberamos el buffer pedido.

# 3. Ejercicio 2

## 3.1. addAndInc

En este ejercicio se debe agregar una palabra pasada por parámetro a la *Distributed HashMap*. Como los sistemas están distribuidos alcanza con que un nodo tome esa palabra y lo agregue en su hashmap local.

### 3.1.1. Consola

Desde el punto de vista de la consola, para realizar esta función, primero debe enviarle un mensaje a todos los nodos disponibles avisándoles que se quiere realizar la función `addAndInc`. Para hacer esto se utiliza la función `MPI_Send` con el tag correspondiente que es 2. El contenido del mensaje que envía no interesa pues es solo un aviso.

A continuación se queda esperando a que todos los nodos le respondan. El remitente del primer mensaje recibido pasa a ser el nodo elegido para realizar la función `addAndInc` y se guarda en la variable `nodoALaburar`.

Luego, se ejecuta un ciclo donde se itera `np` veces y en cada iteración se chequea si el rank del nodo es el que debe realizar `addAndInc`. En ese caso se van a realizar dos `MPI_sends`. El primer send es un aviso para que el nodo se entere de que debe ejecutar `addAndInc`. Esto se realiza mediante el entero `bufi` donde se manda un 1. A continuación se copia la palabra pasada por parámetro en el buffer `buf` y se realiza el segundo send. En caso de que no ser el nodo que tiene que ejecutar `addAndInc` simplemente se envía un mensaje con `bufi` en 0 indicándole que **no** debe ejecutar la función.

Finalmente la consola se queda esperando a recibir un mensaje de todos los nodos mediante la función `MPI_Recv` que indica que los nodos terminaron.

### 3.1.2. Nodo

En esta función el nodo recibe un aviso de que debe realizar `addAndInc`. Luego le envía a la consola un mensaje con la función `MPI_Send`. El contendio del mensaje no interesa pues como ya mencionamos antes la consola se queda esperando una confirmacion de que se va a realizar la funcion.

Luego del send el nodo se queda esperando mediante un `MPI_Recv` a que la consola le avise si debe ejecutar la función o no . El nodo se entera mediante el entero `bufi`. Si este tiene un 1 entonces es el elegido a realizar la función , en caso contrario no debe hacer nada. Cuando al nodo le toca realizar la función este mismo se debe quedar esperando a recibir la palabra que debe agregar en su hashMap. Esto es posible gracias a la función `MPI_Probe` y `MPI_Get_Count` donde en status se captura la palabra junto al tamaño que ocupa. Este tamaño se guarda en una variable llamada `TamMsj`. Luego reasigna el espacio del buffer `buf` acorde al tamaño de la palabra que se envió.

Ahora la función esta lista para realizar un `MPI_Recv` sabiendo que en `buf` se encuentra la palabra. Una vez hecho esto se ejecuta `addAndInc` con el parámetro `buf` de entrada. A continuación se ejecuta `TrabajaArduamente()`.

Al final el nodo debe enviarle un mensaje a la consola para avisarle que termino de realizar el pedido.

# 4. Ejercicio 3

## 4.1. member

La funcion consiste en, pasando una palabra como parametro, identificar si la palabra se encuentra o no en nuestro hashMap.

### 4.1.1. Consola

La funcion de la consola es preguntarle a todos los nodos si tienen a la palabra pasada por parametro. Incluso ya cuando uno responda que si, es necesario esperar a que todos los nodos confirmen si tienen la palabra o no, para garantizar que no quedan mensajes sin leer y que todos los nodos estan disponibles nuevamente.

Para esto, se crea una variable booleana en FALSE. Se pide un buffer donde se guarda la palabra pasada como parametro y se la envia a todos los nodos mediante *MPI_Send* con el TAG de la funcion correspondiente para que el nodo la pueda identificar. Luego esperamos que todos los nodos confirmen si tienen o no la palabra con un ciclo de cantidadDeNodos, utilizando *MPI_Send* esperando cualquier nodo.

### 4.1.2. Nodo

Una vez identificada la funcion member, creamos un buffer del tamaño de un entero. Vamos a utilizarlo para mandar la respuesta.

Con la palabra recibida en el mensaje, aplicamos a nuestro HashMap la funcion $member()$. Luego de trabajar arduamente, verificamos el resultado de la operacion. Si efectivamente la palabra esta en nuestro hashmap, rellenamos nuestro buffer de tamaño entero con un 1. Y si no, con un 0.

Mandamos nuestro buffer mediante un *MPI_Send* a la consola, indicando nuestro resultado.

Finalmente, liberamos la memoria de los buffer.

# 5. Ejercicio 4

## 5.1. maximum

Esta función consulta todos los nodos por su información, de tal manera que luego puede definir cual es la palabra que fue cargada más veces al sistema.

### 5.1.1. Consola

En la consola lo primero que se realiza es crear un Hashmap local, donde agregaremos el contenido de todos los nodos. Luego se envía un mensaje 'dummy' a todos los nodos, el cual no tiene información alguna más que su TAG (4). Utilizando `MPI_Send`, se les informa entonces a los nodos que deben comenzar a enviar el contenido de sus Hashmaps.

Inmediatamente, la consola entra en un ciclo del cual saldrá en cuanto la cantidad de nodos que hayan terminado de enviar sus palabras deje de ser menor a $np - 1$. Dentro del mismo, entonces va recibiendo mensajes que vienen de todos los nodos en simultaneo.

Primero, usando la función `MPI_Probe`, consulta a la estructura $status$ para saber el TAG del mensaje que la consola va a recibir. Si este es 90, eso indica que uno de los nodos ha terminado de enviar su información. Por lo tanto descartamos el mensaje, ejecutando `MPI_Recv`, y aumentamos en 1 el contador de nodos finalizados.

Caso contrario, el TAG del mensaje resulta ser 99, lo cual indicaría entonces que uno de los nodos le esta enviando a la consola una palabra de su Hashmap. Atrapamos este mensaje con `MPI_Recv` dentro de un buffer de caracteres, para luego agregarlo en el Hashmap local con `addAndInc`.

En cuanto todos los nodos han finalizado de enviar sus datos, simplemente se procede a llamar a `maximum` del Hashmap para luego retornar el resultado por consola.

### 5.1.2. Nodo

El nodo por otra parte, se espera a recibir un mensaje de la consola con el TAG 4, el cual cuando llega, este indica que el nodo debe enviar todos sus datos.

Primero descarta el mensaje con `MPI_Recv`, y luego comienza a iterar sobre su Hashmap. Por cada elemento al cual el iterador apunta, este se copia a un buffer de caracteres para después enviárselo de vuelta a la consola vía la función `MPI_Send`.

Cuando el iterador termina de recorrer el Hashmap entero, prosigue a enviar su mensaje de finalización. Para ello simplemente enviamos un mensaje "dummy.ª la consola pero con el TAG fijado en 90, de tal manera que cuando esta lo reciba pueda deducir que un nodo ya no tiene más palabras que enviar.

Finalmente libera todos los recursos que utilizo para comunicarse con la consola antes de volver al ciclo `while(true)`.

# 6. Ejercicio 5

## 6.1. quit

Esta función simplemente debe terminar la ejecución de los procesos.

### 6.1.1. Consola

En la consola se envía un mensaje a todos los nodos para avisarles que deben liberar sus recursos. El contenido del mensaje no importa. Se lo envía mediante `MPI_Send` con el tag correspondiente a quit que es 5.

Luego de realizar el send , se queda esperando a que todos los nodos le avisen que liberaron sus recursos. Esto se hace con la función `MPI_Recv` con el tag `MPI_ANY_SOURCE`. La función receptora esta dentro de un ciclo donde se itera `np` veces para contabilizar la cantidad de mensajes recibidos.

Finalmente, una vez que todos los nodos liberaron sus recursos la consola debe liberar los suyos también.

### 6.1.2. Nodo

El nodo se queda esperando a recibir un mensaje de la consola con `MPI_Recv`. Una vez que lo recibe le envía otro a la consola con `MPI_Send` para avisarle que libero sus recursos. Inmediatamente después del send se liberan los recursos y se sale del ciclo `while(true)` con break para terminar la ejecución.

## 7. Experimentación

Para la experimentación creamos una carpeta bajo el nombre archs. Dentro de esta carpeta se encuentran los siguientes archivos : corpus , corpus-1 , corpus-2 , corpus-3 y zz. A continuación se probaron con los siguientes archivos la funcionalidad de las siguientes funciones :

- **load :** Para verificar la correctitud de esta función ejecutamos el código probando varios $np$ y cantidad de archivos para cargar. Se realizaron las siguientes pruebas :
  - `mpiexec -np 2 ./dist_hashmap` y luego se hizo un `load archs/corpus` : El motivo de esta prueba era para verificar si con un nodo solo se podía cargar un archivo pesado en un tiempo razonable. La lista se proceso instantáneamente.
  - `mpiexec -np 3 ./dist_hashmap` y luego se hizo un `load archs/corpus archs/zz archs/corpus-1 archs/corpus-2 archs/corpus-3` En objetivo de esta prueba era verificar el caso donde se cargan mas archivos que nodos. Los resultados fueron correctos.
  - `mpiexec -np 25 ./dist_hashmap` y luego se ejecuto : `load archs/corpus archs/zz` Los archivos se procesaron correctamente a pesar de haber mas nodos que archivos.
- **addAndInc y member :** Para verificar que estas funciones son correctas se hicieron las siguientes pruebas :
  - `mpiexec -np 5 ./dist_hashmap` y luego se ejecuto `addAndInc pepe`. La palabra se agrego correctamente. A continuación se ejecuto `member pepe` y el resultado fue el esperado, pepe se encuentra en el hashmap.

- `mpiexec -np 5 ./dist_hashmap` y luego se ejecutaron las siguientes instrucciones:
  `load archs/corpus-1`
  `member instrumentos` : Dio que instrumentos estaba , lo cual es correcto.
  `member mate` : Dio que mate no estaba, lo cual es correcto.

- **maximum :** Para esta función se realizaron las siguientes pruebas :

  - `mpiexec -np 5 ./dist_hashmap` y luego se ejecuto `maximum` . El resultado que dio fue la tupla <'',0> que es lo esperado pues los hashMaps están vacíos.

  - `mpiexec -np 10 ./dist_hashmap` y a continuación los siguientes comandos :
    `load archs/corpus-1 archs/corpus-2 archs/corpus-3`
    `maximum`
    El resultado fue <coche,3> que es correcto pues, si bien coche aparece 1 sola vez en cada archivo, en total aparece 3 veces superando las apariciones del resto de las palabras.

  - `mpiexec -np 5 ./dist_hashmap` y a continuación :
    `load archs/corpus-1`
    `maximum`: El resultado fue <instrumentos,2> lo cual es correcto.
    `addAndInc coche`
    `addAndInc coche`
    `maximum` : El resultado fue <coche,3> lo cual es correcto pues supero las apariciones de instrumentos.
    `addAndInc coche`
    `maximum` El resultado fue <coche,4> lo que prueba que el maximum se actualiza correctamente luego de hacer un `addAndInc`.
    `member coche` : Se ejecuto esta función para verificar su funcionalidad luego de haber ejecutado otras funciones previamente. El resultado fue que coche se encontraba en el hashmap lo cual es correcto.