DEPARTAMENTO
DE COMPUTACION

Facultad de Ciencias Exactas y Naturales - UBA

DC - UBA

Sistemas operativos

Trabajo Práctico N°3

Integrante	LU	Correo electrónico
Rodrigo Kapobel	695/12	rok_35@live.com
Esteban Luciano Rey	657/10	estebanlucianorey@gmail.com
Nicolas Hernandez	122/13	nicoh22@gmail.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) 4576-3300
http://www.exactas.uba.ar

Índice

1. Mensajes

Codificación de operaciones

Siendo que una funcionalidad a ejecutar en un nodo se traduce en la llamada a una función, es necesario poder transmitir tanto el nombre de la función a ejecutar como los parámetros a utilizar. Dado que MPI provee el envío de mensajes tagueados (usando el parámetro MPI_Tag en las llamadas a las funciones para identificar un mensaje), esto soluciona el problema de la distinción entre funciones y enviar en el payload los argumentos de las mismas. A continuación se muestran los distintos tags utilizados para mensajes detallados en las descripciones de los métodos:

Nombre de tag	#tag	data adicional	mensajes que lo utilizan
load	1	nombre de archivo	L
addAndInc	2	palabra	A
member	3	palabra	M (member)
maximum	4	null	M (maximum)
quit	5	null	Q (no presente)
ready	6	null	R, EOH (maximum)
id	7	null	Rta A, Winner
word	8	palabra	WordAndCount

Cuadro 1: Codificación de operaciones

Codificación de respuestas

Las respuestas de los nodos a la consola son las siguientes:

- **member**: Los nodos retornan un booleano (True: presente / False: caso contrario)
- **load** : Se notifica a la consola con el mensaje 'ready'cuando el nodo ya procesó un archivo.
- **addAndInc** : Se envía un mensaje ídá la consola para que esta arbitre al ganador.

Para la operación **maximum** se dispuso de una codificación especial. Cada nodo envía por cada palabra diferente, 1 solo mensaje que contiene en la primer sección del buffer la palabra y en los últimos 4 bytes la cantidad de repeticiones de la misma (luego de codificarse como integer)

2. Load

Dada una lista de archivos, el sistema reparte los mismos por disponibilidad de los nodos: se comienza dándole 1 nombre de archivo a cada nodo, y luego a medida que estos finalizan con la carga asignada, solicitan a la consola más trabajo enviando un mensaje con el tag ready.

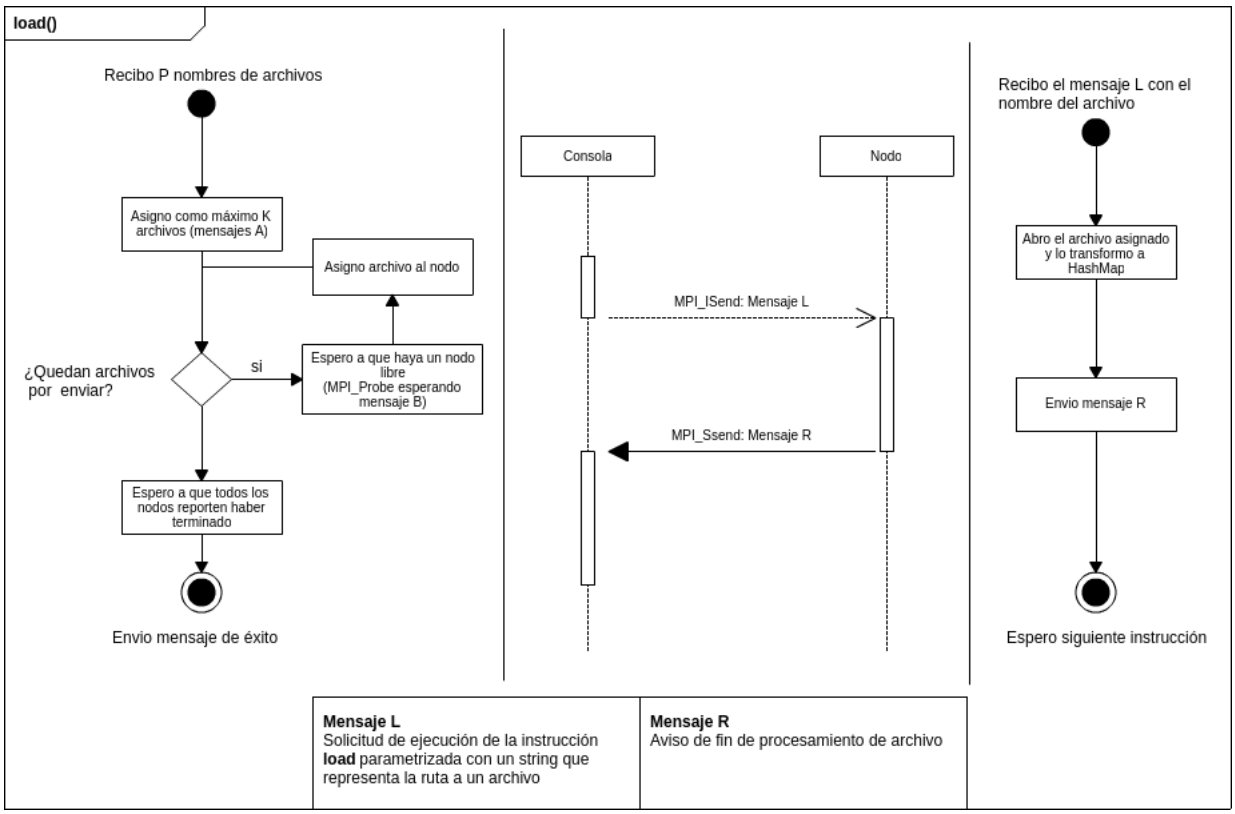

Figura 2.1: Secuencia de eventos de la operación load con K nodos

Para el envío de la instrucción **load** a los nodos se utiliza un mensaje de tipo asincrónico para poder despachar todo el trabajo entre los nodos rápidamente: no hay necesidad de esperar la confirmación de recepción: lo que si interesa es la recepción de la terminación de trabajo, ya que esto determina si la consola puede o no delegar más trabajo en cada nodo: para ello la consola se pone a la escucha de forma bloqueante de mensajes de *cualquier* nodo. Por parte del nodo, como el mismo no recibe ordenes más que de la consola, envía su mensaje **ready** de forma sincrónica, ya que no tiene otra tarea para realizar hasta que la consola sepa que ya se encuentra disponible.

Es importante destacar que el último paso que realizan los nodos es enviar un mensaje **ready** de forma bloqueante. Aquellos nodos que envíen este mensaje y no hayan sido leídos al final de la ejecución del **Load** son leídos para liberar del buffer de mensajes y también liberar a los nodos.

3. addAndInc

Se ingresa una clave al ConcurrentHashMap: para determinar que nodo va a ser el que cargue la palabra se realiza una ”competencia” entre los nodos. La consola elije como ganador al nodo cuya respuesta de participación a la acción de **addAndInc** sea la primera en llegar. Una vez determinado, se les informa a todos los nodos el id del ganador, y solo el nodo con rango igual al id será quien ejecute la operación **addAndInc** ingresando la palabra enviada en el mensaje inicial de **addAndInc**.

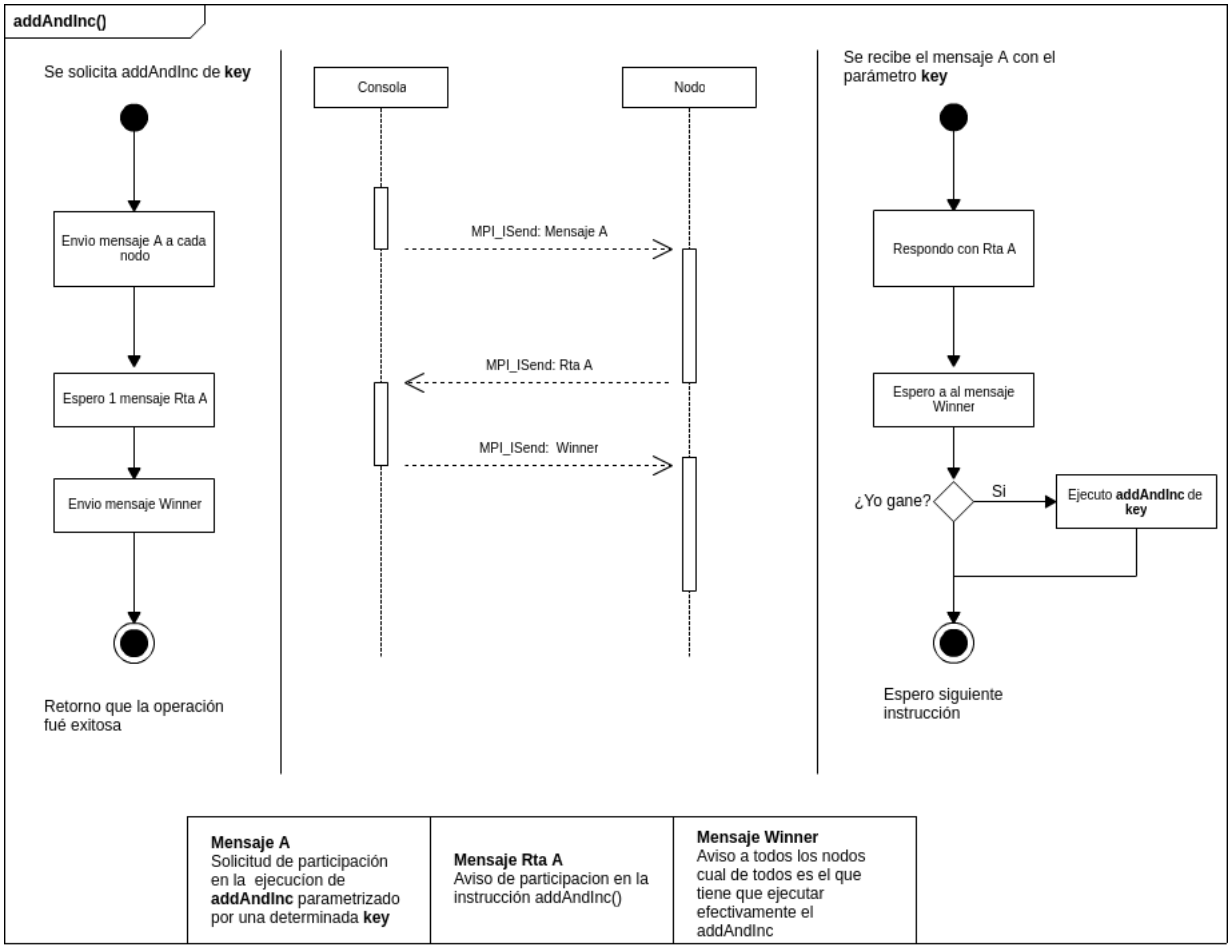

Figura 3.1: Secuencia de eventos de la operación addAndInc

La consola utiliza mensajes asincrónicos para informar del **addAndInc** y la palabra a ingresar. Luego se pone a la escucha de cualquier mensaje para determinar quien será el ganador. Los nodos por su parte, luego de recibir la instrucción de **addAndInc** y guardar temporalmente la palabra a ingresar, se ponen a la escucha de forma bloqueante del mensaje con el ganador. Al recibirlo, en el caso de ser el ganador, ingresa la palabra en su HashMap.

Dado que los nodos envían su mensaje y la consola solo lee uno de ellos, al final de **addAndInc** se libera el buffer de $n-2$ mensajes enviados por los nodos que no fueron leídos (siendo n el total de nodos+consola).

4. Member

La consola envía de forma asíncrona la operación **member** a todos los nodos para que la ejecuten en su HashMap. Los nodos retornan un booleano como respuesta para indicar si la palabra está o no definida en el mapa. La consola ejecuta MPI_Gather para obtener los booleanos que generen los distintos nodos. Los nodos por su parte, luego de realizar la operación, realizan su parte del Gather enviando el booleano para finalizar la transacción.

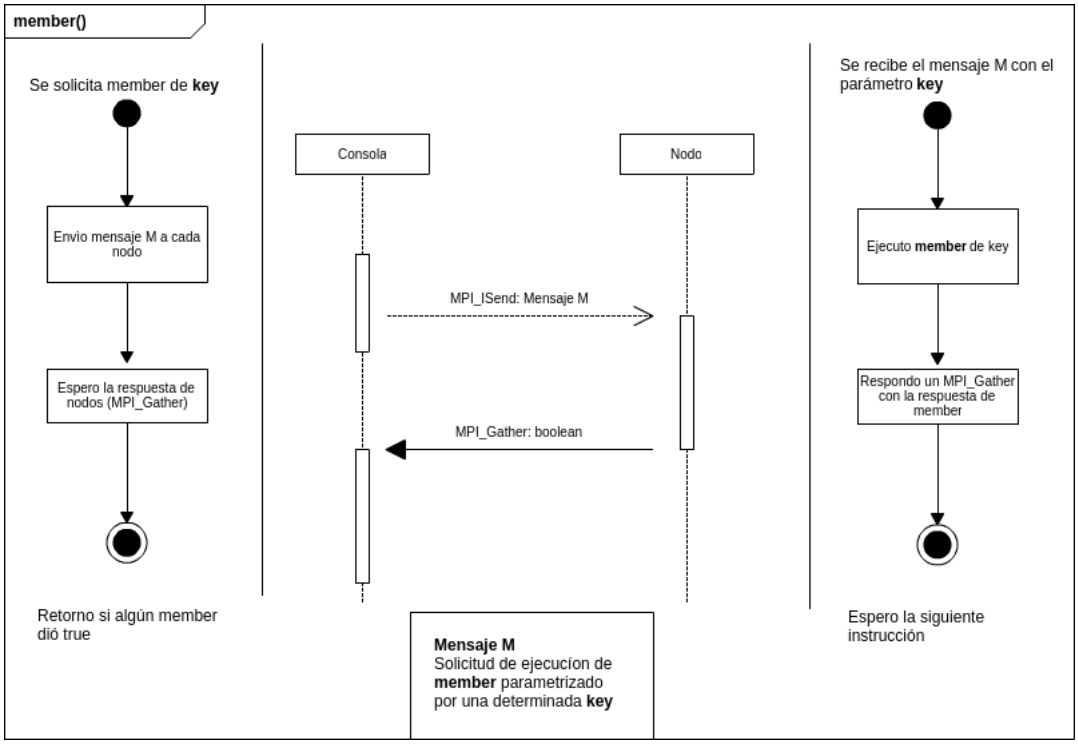

Figura 4.1: Secuencia de eventos de la operación member

5. Maximum

La consola envía la operación **maximum** a todos los nodos de forma asíncrona. Esto genera que los nodos comiencen a mandar de forma sincrónica a la consola 1 mensaje por cada palabra distinta en su HashMap correspondiente a la cantidad de ocurrencias que tienen de la misma. A medida que la consola va recibiendo los mensajes, los almacena en un único HashMap propio. Cuando un nodo finaliza el envío de pares $< valor, repeticiones >$, envía otro con el tag READY para marcar el evento. Cuando el total de finalizados es el total de nodos, el proceso termina y el hashmap general ha sido creado con todas las palabras existentes y sus repeticiones. El último paso es ejecutar la función **maximum**.

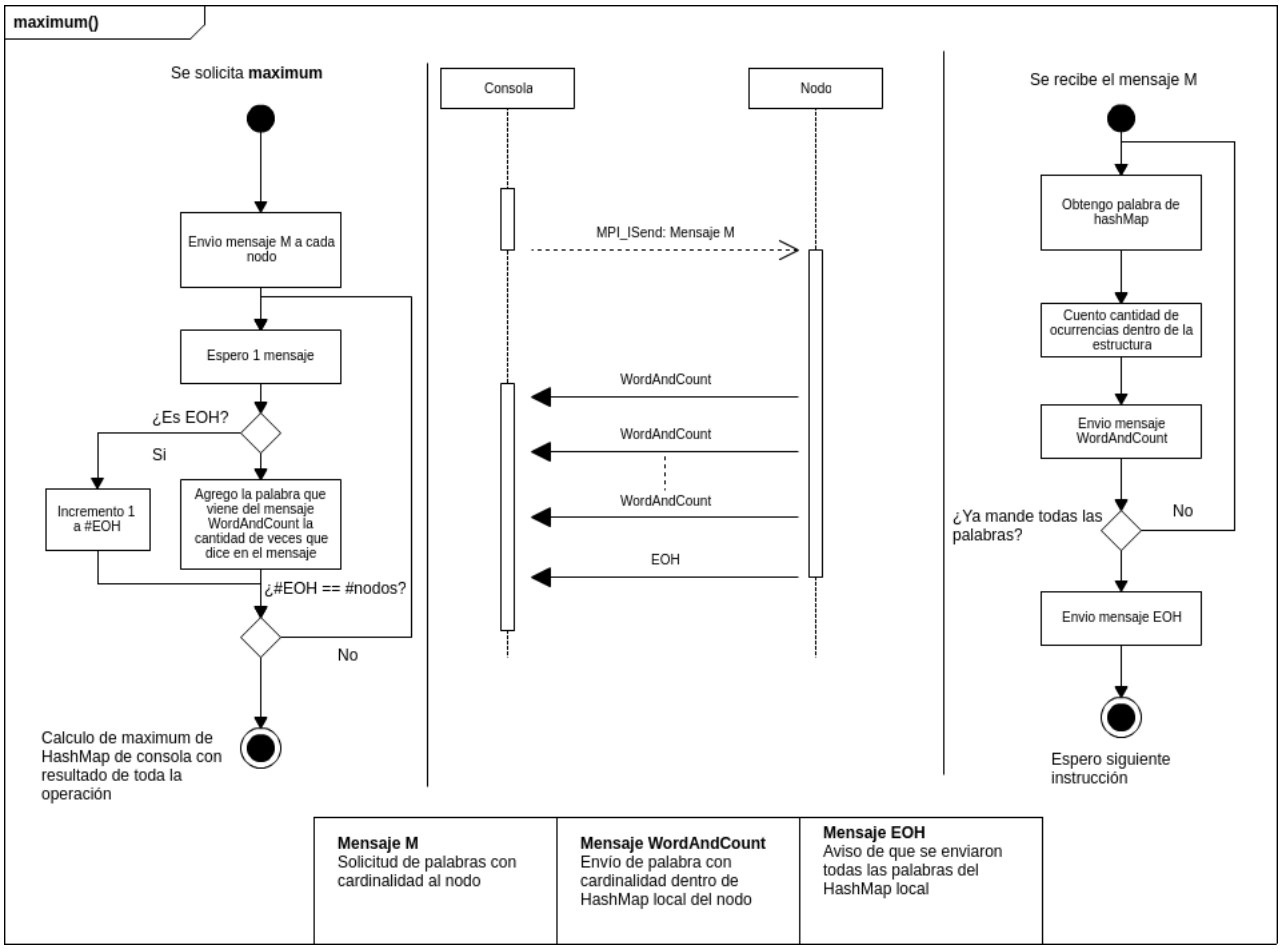

Figura 5.1: Secuencia de eventos de la operación maximum

6. Testing

Para testear el correcto funcionamiento de nuestro codigo generamos 3 sets de archivos diferentes:

1. 5 Archivos de datos *.txt* con las palabras ”perro”, ”auto” y ”gato” en diferentes cantidades.
2. 3 Archivos de instrucciones *.test* que se le pasan al ejecutable *dist_hashmap*.
3. 3 Archivos de comparacion *.expected*, uno por cada *.test*.

Cada archivo *.test* corresponde a un test diferente: Al ejecutarlos se generan archivos *.out* con el output de las instrucciones. Un test se pasa si el *.out* de cada *.test* es igual al correspondiente *.expected*.

Tests

Los tests que ejecutamos son:

- Test 1: Misma cantidad de archivos que nodos. Se corre *mpi* con 6 procesos (una consola y 5 nodos). Luego el test pide que se carguen los 5 archivos y prueba la correctitud de las funciones maximum y member. Luego con addAndInc se agrega una nueva palabra tanta cantidad de veces como sea necesario para que cambie el máximo. Se chequea que en efecto haya cambiado el máximo.
- Test 2: Idem test 1 pero con menos nodos que archivos. Este test nos ayudó a detectar errores en la manera de distribuir los archivos a los nodos.
- Test 3: Idem test 1 pero con menos archivos que nodos.

Ejecución

Los test se corren con la instrucción

```
make test1
make test2
make test3
```