Fº Javier Gutiérrez-Maturana Sánchez

fj.gutierrezs@alumnos.urjc.es

Abstract

Como crear un programa de procesamiento paralelo mediante un entorno cluster en AWS mediante pasos de mesnajes con OpenMPI

Practica de arquitectura de altas prestaciones

Programación paralela mediante paso de mesnajes en un entorno cloud

Índice de Contenidos

[Introducción 2](#_Toc502068305)

[Opciones de diseño utilizadas 2](#_Toc502068306)

[Algoritmo básico 2](#_Toc502068307)

[Control del proceso Master 2](#_Toc502068308)

[Algoritmo de los procesos Map 3](#_Toc502068309)

[Algoritmo de los procesos Reduce 3](#_Toc502068310)

[Algoritmo optimizado 3](#_Toc502068311)

[Control de los procesos Master 3](#_Toc502068312)

[Algoritmo de los procesos map. 4](#_Toc502068313)

[Algoritmo de los procesos reduce 4](#_Toc502068314)

[Cierre del sistema 5](#_Toc502068315)

[Subsistema de comunicaciones de los distintos programas 5](#_Toc502068316)

[Comunicación síncrona entre nodos 6](#_Toc502068317)

[Creación de la arquitectura Cloud 7](#_Toc502068318)

[Administración de las máquinas de un clúster a través de scripting 7](#_Toc502068319)

[Grupo de seguridad 8](#_Toc502068320)

[Conclusiones extraídas 8](#_Toc502068321)

[Tiempo de comunicación 9](#_Toc502068322)

[Comentarios personales 10](#_Toc502068323)

# Introducción

En esta práctica vamos a desarrollar una solución para programación paralela siguiendo un procesamiento paralelo basado en paso de mensajes en un entorno clúster. El programa ejecutará partes del código en nodos o máquinas diferentes, por tanto, no habrá uso de memoria compartida en el programa y la forma de sincronizar y compartir información entre nodos se realiza a través de envío de mensajes.

Nosotros vamos a resolver un problema de computación costoso. Este problema realiza una lectura sobre un conjunto de datos de gran tamaño. El problema no está en la complejidad del algoritmo si no en el conjunto de datos de gran tamaño. Nos fijaremos en si nuestro programa es realmente paralelizadle entre nodos midiendo los tiempos de respuesta cuando aumentamos o disminuimos el número de nodos en un entorno Cloud y realizaremos una medición del Speed Up de nuestro sistema paralelo.

# Opciones de diseño utilizadas

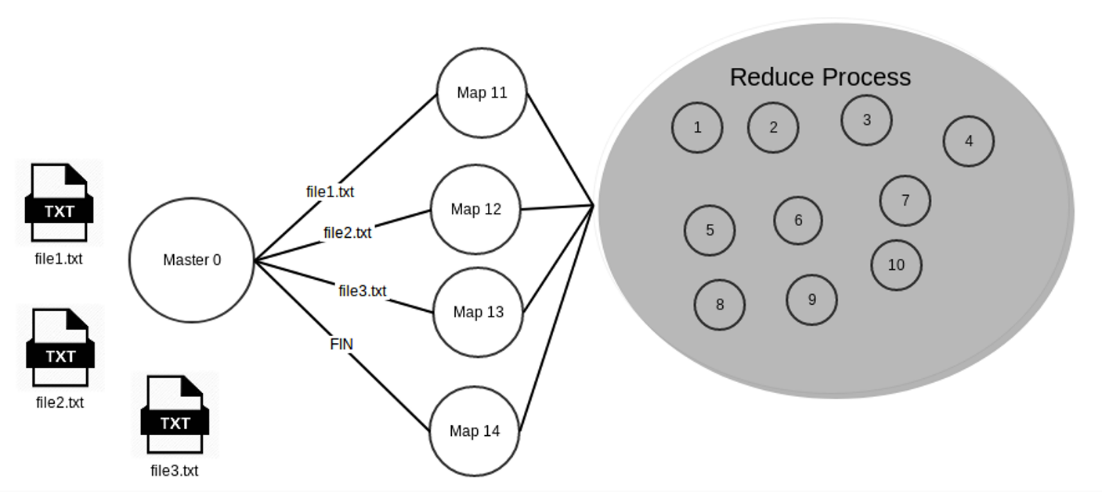
En este apartado se explicarán los diseños utilizados para la realización del algoritmo paralelizable map Reduce en un sistema de máquinas clúster a través de paso de mensajes. Se comentará una primera solución de control de trabajo a través de ficheros y envío de datos de los map y los Reduce y una alternativa de paso de trabajo por lotes para la escalabilidad en los nodos map y envío de datos a los reduce con un menor número de mensajes que mejorará de forma significativa el tiempo de respuesta, por último se comentará el cierre del sistema que es el mismo en ambos diseños.

## Algoritmo básico

Aquí describiremos la primera implementación del algoritmo envío intensivo de mensajes entre nodos y reparto de trabajo a través de ficheros, sin tener en cuenta la cantidad de mensajes o la forma de repartir el trabajo entre los nodos map.

### Control del proceso Master

En un primer diseño se pensó en compartir el trabajo entre los nodos a través de ficheros de texto. El nodo maestro que reparte el trabajo entre los esclavos envía un fichero de texto con un subconjunto de datos a cada nodo map.

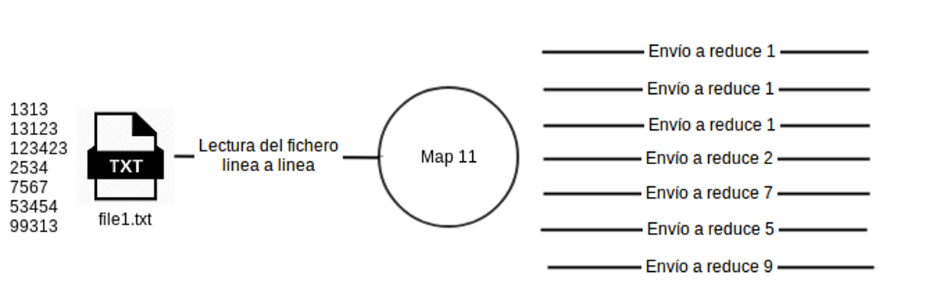


*Figura 1 Esquema de envío de conjunto de datos por fichero.*

El problema de esta implementación es que si se tienen 3 ficheros como en el esquema de ejemplo el nodo maestro mandará la orden de cierre al siguiente map. Solo podemos tener tantos map cómo ficheros, no tendría sentido tener más procesos map que ficheros.

### Algoritmo de los procesos Map

El nodo map lee el fichero completo línea a línea y calcula el primer dígito de cada número de línea. Una vez el proceso a map a leído cuál es el primer número de línea le pasa la línea al nodo reduce que corresponde con el primer dígito.

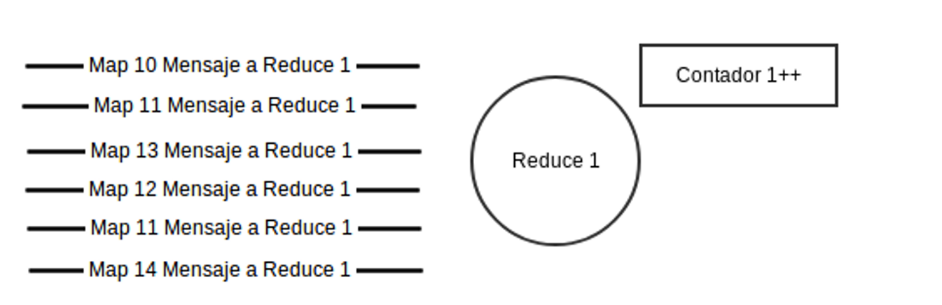


*Figura 2 Esquema de la implementación de los procesos Map.*

El proceso map hará el trabajo más costoso la lectura del conjunto de datos y el cálculo del primer dígito. Una vez sepa cuál es el primer dígito mandará un mensaje al proceso Reduce con el nuevo dígito.

### Algoritmo de los procesos Reduce

Los nodos Reduce están ordenados por su id de 1 a 9. El nodo 1 recibe los números que empiecen por 1 y el nodo 9 recibe los números que empiecen por 9. Los reduce saben cuál es el número que cuentan por su id de proceso. Incrementa el número total de primer dígito por cada mensaje que reciben de 1 en 1 de los nodos map.

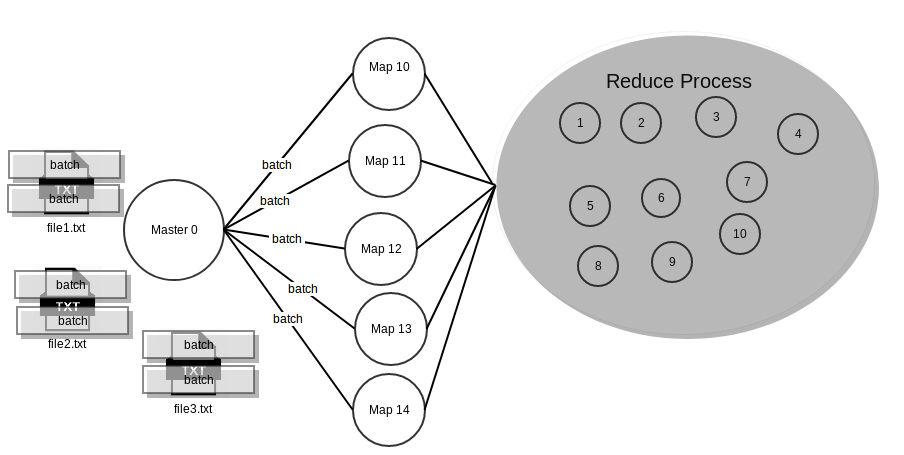


*Figura 3 Diagrama de recibo de mensajes del nodo Reduce 1.*

## Algoritmo optimizado

### Control de los procesos Master

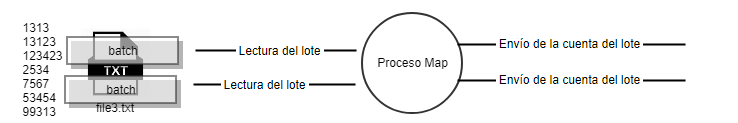
En el primer diseño la lectura de datos reduce el número de procesos map al número de ficheros del conjunto de datos. Si tenemos 4 ficheros solo tendría sentido tener 4 procesos map ya que los siguientes map que creemos serán cerrados por el máster por no tener trabajo para ellos. Para resolver este problema el proceso master cambia el modelo de paso de trabajo a los procesos map, en vez de mandar ficheros completos, el proceso master leerá el tamaño del fichero y el número de procesos map que hay ejecutándose y creará una lectura por lotes para cada map. A cada map le mandará el offset de fichero que debe leer y el tamaño de lote, de esta manera se reparte de forma uniforme el trabajo entre todos los map que compongan el sistema y será mucho más eficiente la lectura de datos por parte del sistema de procesamiento paralelo.



*Figura 4 Esquema del nuevo proceso de paso de trabajo a los map desde el proceso maestro.*

### Algoritmo de los procesos map.

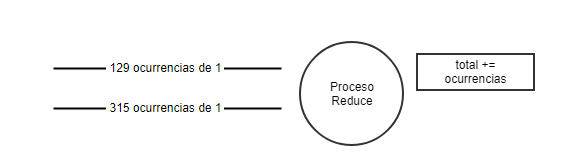
En este caso los nodos map no mandarán un mensaje por cada línea de fichero que lean de su conjunto de datos ya que viendo los tiempos de ejecución con *MPI\_Wtime* ha quedado comprobado que el cuello de botella del sistema está en el paso de mensajes de los procesos map a los procesos reduce. Este es uno de los problemas más importantes a la hora de medir el rendimiento de la primera implementación del sistema. En este algoritmo los nodos map mandarán el total de números leídos al finalizar la lectura completa del lote de datos que le envía el máster.



*Esquema de lectura y paso de mensajes del proceso map*

### Algoritmo de los procesos reduce

Como los procesos map no mandan un mensaje por cada número que leen el proceso reduce no realizará un incremento por mensaje. El proceso reduce leerá el valor del mensaje que será la cantidad total del primer dígito que deberá incrementar a su número total de ocurrencias.



*Esquema de la lógica y paso de mensajes del proceso reduce.*

Por último, realizamos la comprobación del algoritmo de benford con un conjunto de datos obtenidos de internet. Este conjunto de datos cumplía con la ley de benford de los números naturales. El algoritmo de benford se comprobó en el programa final, después de realizar las operaciones map reduce sobre el conjunto de datos se realizaba una comprobación del algoritmo en el nodo master que es el nodo que obtiene los resultados del procesamiento paralelo.

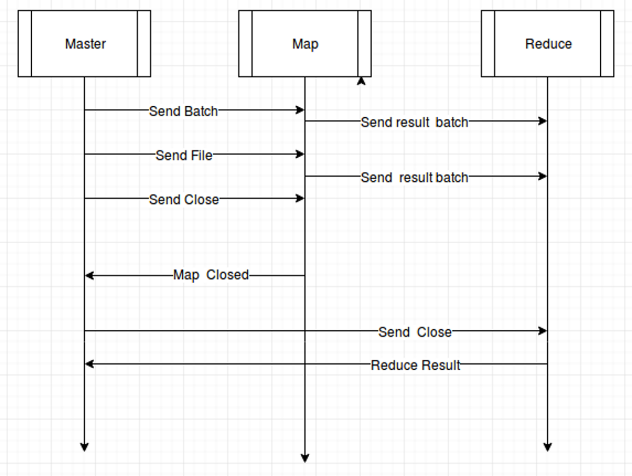
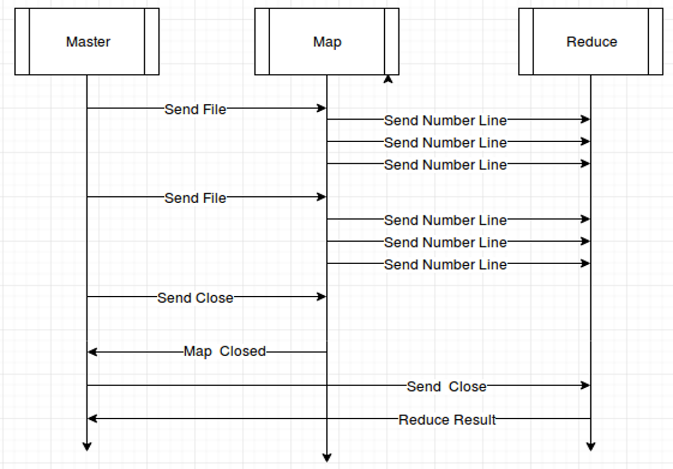
## Cierre del sistema

Una de las cosas más importantes en la lógica de control del máster es la realización del cierre del sistema, por ello explicammos en este apartado cómo se realiza el cierre y la obtención final de resultados.

Cuando el master deja de tener ficheros para los map, manda la señal de fin a los nodos map. Los nodos map terminan de leer y esperan más trabajo por parte del proceso master. Si reciben la señal de *fin* mandan al master una señal de confirmación de que han acabado su ejecución y terminan.

El proceso master una vez manda la señal de fin a los map, se queda esperando a que los map le manden la señal de finalización. Cuando todos los map mandan al master la señal de fin, el proceso master manda el *fin* a los procesos reduce y se queda esperando a los resultados. Los reduce están esperando datos de los map, si se recibe la señal de cierre del proceso master los reduce envían el resultado de cuenta de números al proceso máster y dan por finalizada su tarea.

Una vez los reduce terminan y el proceso master recibe los resultados de los procesos reduce con la suma total de números que recibieron de los map. El proceso máster mostrará una salida con la comprobación del algoritmo de benford y dará por terminada su ejecución.

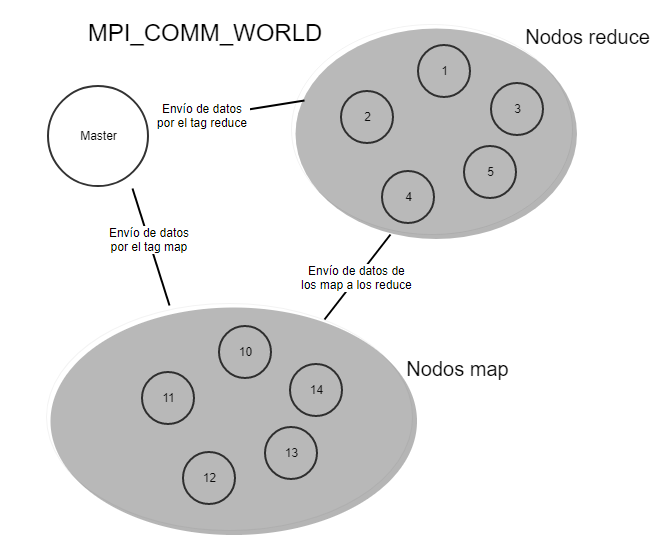


Diagramas UML de ejecución y cierre de los dos sistemas el básico a la izquierda y el optimizado a la derecha.

Se puede comprobar en el diagrama que el número de mensajes intercambiado entre nodos disminuye del sistema inicial al final reduciendo drásticamente el tiempo de respuesta del programa. El cierre del sistema es el mismo en la primera implementación y en la última.

# Subsistema de comunicaciones de los distintos programas

El protocolo de comunicaciones usado para la práctica se explica en el sistema de reparto de trabajo por lotes. Los mensajes de MPI utilizados han sido Send y Recv ambas funciones del framework OpenMPI síncronas. No se ha tenido en cuenta en un primer modelo el uso de Isend y Irecv ya que habría que tener en cuenta el envío asíncrono de los mensajes, la programación de los manejadores y el uso de la llamada Wait que complicaría el diseño, aunque sería mucho más óptimo.



Esquema de red de comunicación entre nodos y diferenciación de los tipos de nodos segmentándolos por etiquetas.

Los nodos distinguirán a los demás nodos en el paso de mensajes por la etiqueta. La etiqueta segmentará la red en los dos tipos de nodos principales, los nodos map y los nodos reduce. Para enviar a un nodo concreto reduce, map o master se deberá definir la etiqueta y el id del nodo.

1. **Comunicación del proceso master:** En el caso de la comunicación con el máster todos los procesos usarán el id del máster y el tag correspondiente al tipo de nodo map o reduce. El master recibirá la información de los nodos distinguiéndolos por el tag asociado al mensaje.
2. **Comunicación con los procesos map**: Los procesos map recibirán datos únicamente del proceso máster y podrán enviar mensajes al nodo máster y a los nodos reduce. El envío de datos a los nodos reduce se realizará por su id y la etiqueta reduce.
3. **Comunicación con los procesos reduce:** Los nodos reduce recibirán datos del nodo máster y de los nodos map y solo podrán mandar mensajes al nodo máster con la suma de resultados del número obtenido. Distinguirán al nodo máster por la señal de cierre que será un número de ocurrencias negativo.

## Comunicación síncrona entre nodos

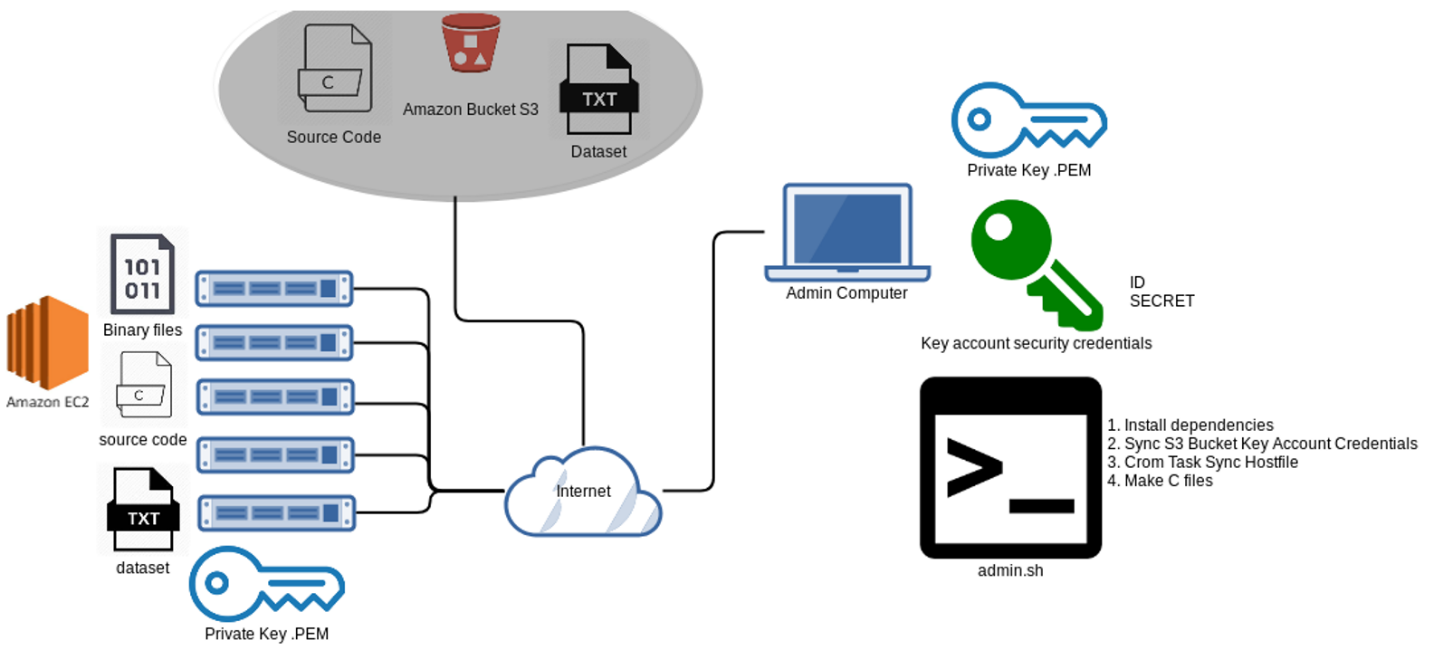
La comunicación es completamente síncrona ya que al ser una primera implementación del algoritmo paralelo con paso de mensajes se debe realizar de manera adecuada tanto el envío de trabajo como el cierre del sistema evitando los interbloqueos y mostrando una solución lo más general y simple posible. Una vez tenemos el sistema implementado de manera correcta de forma síncrona podemos pensar en un modelo asíncrono que realizará el trabajo de manera más óptima, pero será más complejo de implementar. También deberíamos pensar cómo disminuir aún más el número de paso de mensajes entre los nodos y mejorar la gestión del envío y recibo de mensajes

# Creación de la arquitectura Cloud

En este apartado explicaremos cómo hemos realizado la creación del clúster de máquinas virtuales en AWS. Estas máquinas serán los diferentes nodos que ejecutarán el código del programa paralelo con OpenMPI. Hablaremos de cómo realizar la administración de las máquinas a través de scripting y la sincronización de ficheros a través de S3. También las medidas de seguridad necesarias para el uso programático de los servicios de Amazon Web Services y control de acceso a las instancias y las reglas de seguridad del firewall para permitir la comunicación entre nodos con OpenMPI.

## Administración de las máquinas de un clúster a través de scripting

La administración de máquinas puede resultar muy compleja en clústeres de ordenadores si no se tiene en cuenta la escalabilidad desde el inicio. El administrador debe ser capaz de manejar un conjunto N de máquinas muy alto de manera sencilla y en el menor tiempo posible. En esta práctica se ha tenido en cuenta la labor administrativa de máquinas en el Cloud y se ha creado un script de administración para la configuración inicial de las máquinas virtuales de AWS.

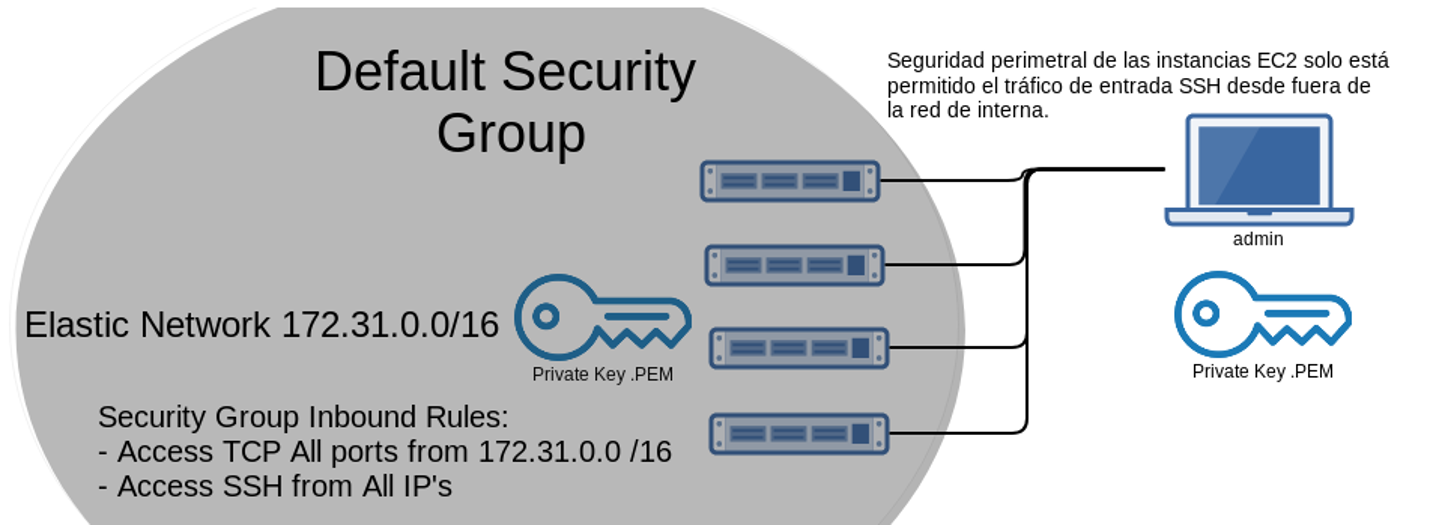
*Figura 1 Esquema de configuración del sistema de administración del clúster.*

Explicaremos uno a uno los componentes del clúster y su función dentro de la práctica.

1. **Clave de seguridad de credenciales de cuenta:** Esta clave permite acceder a los servicios de Amazon de forma programática es importante no compartir esta clave con nadie ni dejarlas en sitios públicos. Está compuesta por el ID y un secreto ambos son cadenas de caracteres que identifican de manera unívoca la clave de credenciales del usuario. La clave tendrá asociada las autorizaciones a los servicios del Cloud que se pueda acceder con ella. En nuestro caso la clave debe tener acceso al servicio de S3 para realizar la sincronización desde el script de administración en las máquinas EC2 del código del programa.
2. **Private Key PEM:** Esta clave privada es la que usaremos para conectar las máquinas por SSH desde la máquina de administración. Todas las instancias EC2 tendrán esta clave ya que será necesaria para acceder con OpenMPI para realizar el paso de mensajes entre las máquinas virtuales del clúster.
3. **admin.sh:** Este es un script de administración que se ejecutará al iniciar las máquinas EC2 del clúster instalará las dependencias OpenMPI y la consola de línea de comandos awscli. Después realizará la bajada de los archivos del código fuente y los conjuntos de datos de S3 a través de la clave de credenciales que hemos creado de nuestra cuenta. Por último creará una tarea programada con cron para actualizar el fichero hostfile en las máquinas EC2 que se encuentre en S3. Este fichero hostfile se obtendrá a través de una petición wget por tanto el fichero hostfile debe ser público en nuestro bucket de S3.
4. **Instancias EC2:** Las instancias EC2 son las máquinas virtuales que usaremos para componer el clúster y la realización de las medidas del procesamiento paralelo con Map Reduce. Se conectarán entre ellas y todas tendrán el conjunto de datos replicado en su soporte de almacenamiento para paralelizar la lectura del conjunto de datos.

## Grupo de seguridad

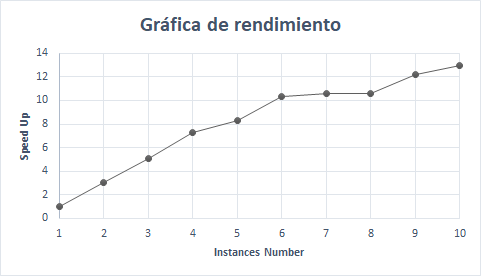
Amazon EC2 debe configurarse con un grupo de seguridad. Este grupo de seguridad define las reglas Firewall de entrada y salida de comunicaciones en las instancias EC2. Deberemos realizar una configuración del firewall a través de un grupo de seguridad que será común para todas las instancias EC2. De esta manera el programa Map Reduce podrá acceder a las demás instancias y realizar el paso de mensajes con OpenMPI.

*Esquema de grupo de seguridad de las instancias EC2 para el paso de mensajes en OpenMPI.*

Las instancias tendrán acceso del tráfico TCP de la red interna *Elastic Network* y se podrá acceder a través de internet al servicio SSH para el acceso de administración a través de la clave privada que comparten todas las instancias EC2 y la máquina de adminstración.

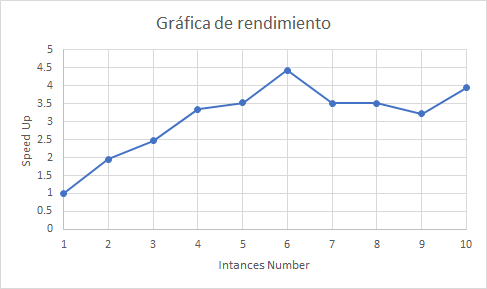
# Conclusiones extraídas

Hemos comprobado que el número de mensajes que se envían entre los nodos es un problema vital a la hora de realizar el sistema paralelo. Si el sistema pasa un número de mensajes excesivo entre los nodos se puede ver afectado el rendimiento del sistema de manera realmente considerable. En la primera parte de la práctica el tiempo de ejecución más pequeño que teníamos era de 32 segundos con el uso de 10 instancias. Cuando bajamos el número de mensajes de comunicación entre nodos, el tiempo de ejecución del mismo conjunto de datos no tardaba más de 0.5 segundos en una única instancia. Esto podía verse como un aumento del Speed Up en 828. Es decir hemos mejorado el algoritmo con el decremento del paso de mensajes en 828 unidades. Nuestro sistema es 828 veces más rápido. Para sacar el tiempo de rendimiento hemos usado el comando time y el comando *MPI\_Wtime* para medir el tiempo de forma más específica en cada nodo y comprobar que partes del código dan mayores problemas de lentitud, en la mayoría de casos el mayor tiempo de ejecución se conentra en las llamas MPI de envío y recibo de mensajes del código del programa y la mayor parte del tiempo de ejecución se conetra en la lectura de los lotes de los procesos map y el envío de mensajes a los procesos reduce.



*Gráficas de rendimiento del algoritmo básico con un mismo conjunto de datos. Aumento del número de instancias.*

El sistema básico funcionaba lento, el uso de más instancias a modo clúster con el programa realizaba una mejora significativa en el Speed up de las máquinas, siguiendo la gráfica la ley de Gustafson, se ejecutan las tareas con gran peso computacional en varias máquinas de manera paralela incrementando el tiempo de respuesta de manera lineal cuando aumentamos el número de máquinas.



*Gráfica de rendimiento del algoritmo optimizado con un conjunto de datos alto.*

En la segunda parte de resultados se comprueba el Speed up respecto al segundo algoritmo, el sistema basado en lotes optimizado. Este segundo sistema cumple la ley de Gustafson hasta que el trabajo ya no es significativo. El conjunto de datos no es lo suficientemente alto para mejorar en el tiempo de respuesta del programa, aunque aumentemos el número de máquinas.

## Tiempo de comunicación

La mayor parte del tiempo de respuesta de un programa paralelizable se pierde en el tiempo de comunicación entre nodos, sobre todo en la latencia, tiempo que tarda en llegar el primer bit a destino. En este apartado hemos puesto el porcentaje de tiempo que tardamos en el envío y recibo de mensajes respecto al tiempo total de ejecución, . Hemos usado la función de OpenMPI *MPI\_Wtime* para medir el tiempo de envío y recibo de mensajes en las partes del código donde se realizan estas acciones y hemos medido el tiempo total de respuesta del programa para medir cuanto tiempo se pierde en la comunicación entre nodos.

Comprobando el tiempo de procesamiento medio por lote de los nodos map que está en una media de *200ms* el mayor tiempo de ejecución del nodo maestro está en el envío de mensajes a los nodos map y la espera de respuesta de cierre de los map y reduce. Los reduce tiene un tiempo de bloqueo po comunicación alto debido a que están esperando la mayor parte del tiempo datos de los nodos map para realizar la suma total de ocurrencias del número que gestionan.

## Comentarios personales

No ha resultado difícil de gran complejidad la realización de sistema veníamos con una base de programación de comunicaciones del grado de Telemática y se sabía que una de las partes más problemáticas era la lógica del nodo máster, como se realizaba el envío de mensajes entre nodos y el cierre del sistema. Además de la correcta gestión de la memoria en el uso del lenguaje de programación C que se vio en sistemas operativos en la carrera.

La parte de subsistemas de comunicaciones es mejorable con el paso de mensajes asíncronos entre nodos, una mejor gestión de las etiquetas del comunicador y los id's de los nodos y una reducción del número de mensajes entre nodos. La realización de evitar los bloqueos en el nodo maestro puede mejorar el tiempo de respuesta, el bloqueo de los nodos reduce es inevitable debido que no pueden realizar operaciones sin datos de los nodos map.

La implementación optimizada dio muy buenos resultados en el tiempo de respuesta final de la práctica reduciendo de manera muy significativa el tiempo de ejecución del programa y repartiendo el trabajo entre un mayor número de procesos map.

No encontramos realmente cambios en el tiempo de respuesta hasta que no usamos un conjunto de datos realmente alto en el algoritmo optimizado donde la mayor parte del tiempo se realiza en la lectura de datos de los nodos map.

La mayor parte del objetivo del algoritmo map reduce es el reparto de trabajo entre los nodos para incrementar la respuesta del sistema, pero si este conjunto no era lo suficientemente alto el tiempo de respuesta no variaba de manera significativa en el incremento de los nodos, como se pudo ver en la segunda gráfica de medidas de rendimiento del programa.