Modelos de programación y cargas de trabajo para computadores escalables

UNIVERSIDAD INDUSTRIAL DE SANTADER
Fabian León: 2140180 Fabio Geney: 2140178.

***Resumen*—Las computadoras escalables son sistemas que han tomado gran importancia en los últimos años, debido a su capacidad de procesar grandes cantidades de datos de orden de hasta peta-bytes. estos sistemas cuentan con una arquitectura y operación diferente a los centros de datos, ya que permiten un amplio nivel de paralelismo y usar frameworks como MapReduce y su version libre Hadoop.**

***palabras clave*—WSC, MapReduce, Hadoop, computadoras escalables, paralelismo.**

***abstract*—Scalable computers are systems that have taken on great importance in recent years, because of their ability to process large amounts of data from order to peta-bytes. These systems feature an architecture and a different operation than data centers because they allow for a wide level of parallelism and use frames like MapReduce and its free Hadoop version.**

***key words*—WSC, MapReduce, Hadoop, Scale Computing, Scale Computing.**

I. Introducción

El gran crecimiento y popularidad que ha tomado los servicios de Internet como búsqueda, redes sociales, mapas en linea, compras en linea, correo electrónico entre otros ha hecho que se piense en nuevas arquitecturas y modelos de programación para manejar de forma eficaz y rápida la gran cantidad de datos que se generan diariamente y que cada vez crecen mas, pensando en esto se vio la necesidad de crear Warehouse scale-computers(WSC)(considerados los descendientes modernos de las supercomputadoras) pareciera que fueran simples centros de datos pero estos cuentan con una arquitectura y forma de operar diferente. Estos sistemas cada vez toman mas popularidad por su paralelismo y uso de frameworks para procesar grandes lotes de datos, se puede imaginar que los WSC es una tecnología parecida a la computación de alto rendimiento(HPC)[1] pero los WSC difieren de los HPC por su bajo costo de implementación haciéndolo mas asequible para el mundo, hoy en día la demanda de WSC es mayor que la de HPC.

[1]Esta es una tecnología que permite enlazar servidores en un cluster de alto poder para realizar computación paralela.

II. Modelos de programación

Son las características que expresan la estructura lógica de un programa entre estas están: complejidad, rendimiento, costo de mantenimiento, paralelismo entre otros. Los principales tipos de programación son:

II-A. Programación lineal

Es principal mente usada para maximizar o minimizar funciones lineales, en especial sistemas de ecuaciones o inecuaciones igualmente lineales.

La programación lineal se desarrolló como modelo matemático para planificar costos y retornos, fue usada principalmente en sus inicios dentro de la segunda guerra mundial para reducir costos militares. Varios fueron los fundadores de estas técnicas matemáticas, entre los cuales se encuentran, Dantzig, quien publicó el algoritmo Simplex, Khachiyan que diseñó el llamado algoritmo del elipsoide, a través del cual demostró que el problema de programación lineal puede ser resoluble de manera eficiente, es decir, en tiempo polinomial.

El siguiente es un ejemplo extraído de wikipedia; En este ejemplo se tratan solamente 6 variables, pero un problema de este tipo puede tener fácilmente alrededor de 1000 variables, se usa el algoritmo simplex para reducir la solución como se muestra en la figura (1).

Existen tres minas de carbón cuya producción diaria es:

- La mina "a" produce 40 toneladas de carbón por día;
- La mina "b" produce 40 t/día; y,
- La mina "c" produce 20 t/día.

En la zona hay dos centrales termoeléctricas que consumen:

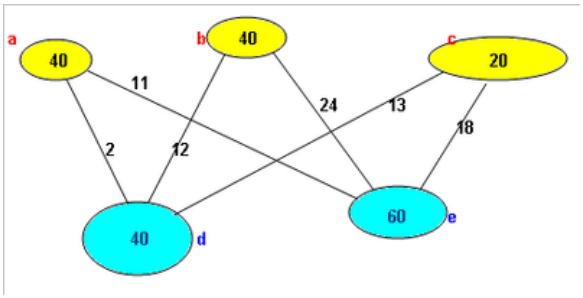

Figura 1. Algoritmo simplex: Es un método analítico de solución de problemas de programación lineal capaz de resolver modelos mas complejos que los resueltos por métodos gráficos sin restricción de numero de variables.

- La central "d" consume 40 t/día de carbón; y,
- La central "e" consume 60 t/día.

Los costos de mercado, de transporte por tonelada son:

- De "a" a "d" = 2 monedas
- De "a" a "e" = 11 monedas
- De "b" a "d" = 12 monedas
- De "b" a "e" = 24 monedas
- De "c" a "d" = 13 monedas
- De "c" a "e" = 18 monedas

Si se preguntase a los pobladores de la zona cómo organizar el transporte, tal vez la mayoría opinaría que debe aprovecharse el precio ofrecido por el transportista que va de "a" a "d", porque es más conveniente que los otros, debido a que es el de más bajo precio.
En este caso, el costo total del transporte es:

- Transporte de 40 t de "a" a "d" = 80 monedas
- Transporte de 20 t de "c" a "e" = 360 monedas
- Transporte de 40 t de "b" a "e" = 960 monedas
- Total 1.400 monedas.

Sin embargo, formulando el problema para ser resuelto por la programación lineal se tienen las siguientes ecuaciones:

- Restricciones de la producción:

$$X_{a\to b} + X_{a\to e} \leq 40[T/dia]$$
$$X_{b\to d} + X_{b\to e} \leq 40[T/dia]$$
$$X_{c\to d} + X_{c\to e} \leq 20[T/dia]$$

- Restricciones del consumo:

$$X_{a\to d} + X_{b\to d} + X_{c\to d} \geq 40[T/dia]$$
$$X_{a\to e} + X_{b\to e} + X_{c\to e} \geq 60[T/dia]$$

- La función objetivo será:

$$2X_{a\to d} + 11X_{a\to e} + 12X_{b\to d} + 24X_{b\to e} + 13X_{c\to d} + 18X_{c\to e} = Min!$$

II-B. Programación paralela

Es una forma de computación en la cual varios cálculos pueden realizarse simultáneamente, basado en el principio de dividir los problemas grandes para obtener varios problemas pequeños, que son posteriormente solucionados en paralelo. Hay varios tipos diferentes de paralelismo: nivel de bit, nivel de instrucción, de datos y de tarea.

La programacion en paralelo cada vez se hace mas famosa y la programacion secuencial[2] se ve cada vez mas saturada debido a la forma en la que ha crecido la complejidad de algoritmos y la cantidad de datos que se deben procesar.

Las ventajas de la programación en paralelo es grande sobre todo en la optimización del trabajo, a pesar de que no todos los problemas pueden implementarse en paralelo por la dependencia de datos, en general la idea de dividir problemas grandes en problemas mas pequeños para que varias máquinas resuelvan un problema es de gran ayuda.

II-B1. Paralelismo a nivel de bits: El aumento del tamaño de la palabra reduce el número de instrucciones que el procesador debe ejecutar para realizar una operación en variables cuyos tamaños son mayores que la longitud de la palabra.

II-B2. Paralelismo a nivel de instrucciones: El paralelismo a nivel de instrucción consiste en un técnica que busca que la combinación de instrucciones de bajo nivel que ejecuta un procesador puedan ser ordenadas de forma tal que al ser procesadas en simultáneo no afecten el resultado final del programa, y más bien incrementen la velocidad y aprovechen al máximo las capacidades del hardware. Un pipeline (canalizador) de instrucciones es el que permite que por cada ciclo de reloj del procesador múltiples instrucciones se encuentren en distintas fases de ejecución.

- Pipeline: Prácticamente todos los procesadores, ya sea que se encuentren en supercomputadoras de alto rendimiento, máquinas de escritorio, consolas de videojuego, sistemas empotrados, etc., utilizan un pipeline de instrucciones. En el mismo puede obtenerse una nueva instrucción en cada ciclo de

[2]son problemas donde solo participan operaciones, entradas y salidas

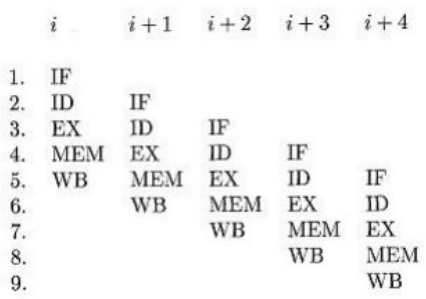

Figura 2. Pipeline de 5 etapas ejecutando 5 instrucciones consecutiva.
IF: Instruction Fetch: obtiene la instrucción.
ID: Instruction Decode: decodifica la instrucción.
EX: Execute: ejecuta la operación.
MEM: Memory: accede la memoria (lecturas y escrituras)
WB: Write Back: escribe el resultado con lo que se finaliza la ejecución de instrucción.

reloj, mientras que las siguientes instrucciones pueden seguir ingresando conforme el reloj avanza. En la Figura 2 se muestra un pipeline de instrucciones simples compuesto por 5 etapas:
La figura muestra cómo las instrucciones i, i + 1, i + 2, i + 3, i + 4 pueden ser ejecutadas al mismo tiempo. Si el resultado de una instrucción se encuentra disponible, y una instrucción siguiente necesita de ellos para ser ejecutada, el procesador será entonces capaz de emitir una instrucción cada ciclo de reloj.

II-B3. Paralelismo a nivel de datos: Es un paradigma de la programación concurrente que consiste en subdividir el conjunto de datos de entrada a un programa, de manera que a cada procesador le corresponda un subconjunto de esos datos. Cada procesador efectuará la misma secuencia de operaciones que los otros procesadores sobre su subconjunto de datos asignado. En resumen: se distribuyen los datos y se replican las tareas.

Idealmente, esta ejecución simultánea de operaciones, resulta en una aceleración neta global del cómputo.

El paralelismo de datos es un paradigma suficientemente adecuado para operaciones sobre vectores y matrices, dado que muchas de ellas consisten en aplicar la misma operación sobre cada uno de sus elementos.

II-B4. Paralelismo a nivel de tareas: es un paradigma de la programación concurrente que consiste en asignar distintas tareas a cada uno de los procesadores de un sistema de cómputo. En consecuencia, cada procesador efectuará su propia secuencia de operaciones.

En su modo más general, el paralelismo de tareas se representa mediante un grafo de tareas, el cual es subdividido en subgrafos que son luego asignados a diferentes procesadores. De la forma como se corte el grafo, depende la eficiencia de paralelismo resultante. La partición y asignación óptima de un grafo de tareas para ejecución concurrente es un problema NP-completo, por lo cual en la práctica se dispone de métodos heurísticos aproximados para lograr una asignación cercana a la óptima.

Sin embargo, existen ejemplos de paralelismo de tareas restringido que son de interés en programación concurrente. Tal es el caso del paralelismo encauzado, en el cual el grafo tiene forma de cadena, donde cada nodo recibe datos del nodo previo y sus resultados son enviados al nodo siguiente. El carácter simplificado de este modelo permite obtener paralelismo de eficiencia óptima

III. ESCALABILIDAD

Es la propiedad deseable de un sistema, una red o un proceso, que indica su habilidad para reaccionar y adaptarse sin perder calidad, o bien manejar el crecimiento continuo de trabajo de manera fluida, o bien para estar preparado para hacerse más grande sin perder calidad en los servicios ofrecidos.

En general, también se podría definir como la capacidad del sistema informático de cambiar su tamaño o configuración para adaptarse a las circunstancias cambiantes.

La escalabilidad como propiedad de los sistemas es generalmente difícil de definir, en particular es necesario definir los requisitos específicos para la escalabilidad en esas dimensiones donde se crea que son importantes. Es una edición altamente significativa en sistemas electrónicos, bases de datos, ruteadores y redes. A un sistema cuyo rendimiento es mejorado después de haberle añadido más capacidad hardware, proporcionalmente a la capacidad añadida, se dice que pasa a ser un sistema escalable.

Hay dos tipos básicos de escalabilidad, en base al método que se usa para aumentar la capacidad de un sistema:

III-1. Escalabilidad vertical: Se refiere a actualizaciones o modernización de componentes existentes, por ejemplo aumentar el número de CPUs que tiene el servidor de un sitio web. Si un programa mejora al aumentar los recursos que puede utilizar, se dice que es escalable, a lo que se conoce también como escalabilidad de aplicación.

III-2. Escalabilidad horizontal: Se refiere a aumentar el número de componentes, para usar el mismo ejemplo, en lugar de aumentar el número de CPUs, se aumenta el número de computadoras que sirven un sitio web. Es común que un sistema tenga definido un número máximo de computadoras a as que se puede escalar, a lo que se conoce como escalabilidad de tamaño.

IV. Balanceo de carga

el balanceo de carga es la manera en que las peticiones de Internet son distribuidas sobre una fila de servidores. Existen varios métodos para realizar el balanceo de carga. Desde el simple "Round Robin"[3] hasta los equipos que reciben las peticiones, recogen información, en tiempo real, de la capacidad operativa de los equipos y la utilizan para enrutar dichas peticiones individualmente al servidor que se encuentre en mejor disposición de prestar el servicio adecuado.

Los balanceadores de carga pueden ser soluciones hardware, tales como routers y switches que incluyen software de balanceo de carga preparado para ello, y soluciones software que se instalan en el back end de los servidores.

V. Warehouse-scale computer(WSC)

los WSC es la fundación de los servicios de Internet que mucha gente utiliza diario; Búsqueda, redes sociales, mapas en línea, compartición de vídeo, compras en línea, servicios de correo electrónico, entre otros. La tremenda popularidad de tales servicios de Internet hizo necesaria la creación de WSCs que pudieran mantenerse al día con las rápidas demandas del público. Hoy en día, los WSC actúan como una máquina gigante y cuestan del orden de $150 millones de dolares para la construcción , la infraestructura eléctrica y de enfriamiento, los servidores y el equipo de redes que conecta y aloja de 50.000 a 100.000 servidores.

[3]Es un método donde Las peticiones clientes son distribuidas equitativamente entre todos los servidores existentes

La gran diferencia entre WSC y los demás datacenters es que, actualmente WSC funciona como una sola máquina y, disponible para todo el mundo.

los WSC y los servidores comparten algunas características y requerimientos tales como:

- Desempeño-costo: El trabajo hecho por dólar es crítico en parte por la escala. Reducir el costo de capital de un WSCs en un 10 % podría ahorrar $15M.
- Eficiencia energética: la eficiencia energética es de bastante importancia debido que de esta depende también el sistema de enfriamiento; los picos de energía y la energía consumida conducen el coste de la distribución de energía y el coste de sistemas de enfriamiento. Además, la eficiencia energética es una parte importante de la gestión ambiental. Por lo tanto, el trabajo realizado por joule es crítico tanto para WSCs como para servidores debido al alto costo de construcción de la infraestructura eléctrica y mecánica para un almacén de computadoras y para las facturas mensuales de servicios públicos a los servidores de energía.
- Fiabilidad a través de la redundancia: la naturaleza de larga duración de los servicios de Internet significa que el hardware y el software en una WSC deben proporcionar colectivamente al menos el 99,99 % de la disponibilidad; Es decir, debe estar abajo menos de 1 hora por año. La redundancia es la clave para la confiabilidad tanto para WSCs como para servidores. Mientras que los arquitectos de servidores a menudo utilizan más hardware ofrecido a un costo más alto para alcanzar una alta disponibilidad, los arquitectos de WSC confían en múltiples servidores rentables conectados por una red de bajo costo. Múltiples WSC también reducen la latencia para los servicios que están ampliamente desplegados.
- red de entrada y salida:los arquitectos de servidores deben proporcionar una buena interfaz de red al mundo externo, y los arquitectos de WSC también deben hacerlo. Es necesario establecer redes para mantener la coherencia de los datos entre varias WSC, así como para establecer una interfaz con el público.

los WSC tambien tienen características y requerimientos que no comparten con los servidores como:

- Amplio paralelismo: una preocupación para un arquitecto de servidor es si las aplicaciones en el lugar de destino tienen suficiente paralelismo para

justificar la cantidad de hardware paralelo. Un arquitecto de WSC no tiene tal preocupación. En primer lugar, las aplicaciones de lote se benefician del gran número de conjuntos de datos independientes que requieren un procesamiento independiente, como miles de millones de páginas web de un rastreo web. Este procesamiento es paralelismo de nivel de datos aplicado a los datos en el almacenamiento en lugar de los datos en la memoria. En segundo lugar, las aplicaciones interactivas de servicios de Internet, también conocidas como software como servicio (SaaS), pueden beneficiarse de millones de usuarios independientes de servicios interactivos de Internet. Lecturas y escrituras rara vez dependen de SaaS, por lo que SaaS raramente necesita sincronizarse. Por ejemplo, la búsqueda utiliza un índice de sólo lectura y el correo electrónico suele leer y escribir información independiente.

- costo operativo: los arquitectos de servidores usualmente diseñan su sistema para obtener un rendimiento máximo dentro de un presupuesto de costos y se preocupan por la energía sólo para asegurarse de que no superen la capacidad de refrigeración. Las WSC tienen una vida útil más larga, la infraestructura eléctrica y la infraestructura de refrigeración se estima que dura al rededor de 10 años o mas, pero se le agrega el costo de operaciones como: energía, distribución de energía y refrigeración representando más del 30 % de los costos de una WSC en 10 años

los precursores de las WSC son clusters[4]; Para cargas de trabajo que no requerían una comunicación intensiva, los clusters ofrecían una computación mucho más efectiva que los multiprocesadores de memoria compartida. Una visión de WSCs es que son sólo la evolución lógica de cluster de cientos de servidores a decenas de miles de servidores. Una cuestión natural es si las WSC son similares a los clusters modernos para la computación de alto rendimiento. Aunque algunos tienen escala y costo similares hay diseños HPC con un millón de procesadores que cuestan cientos de millones de dólares que generalmente tienen procesadores mucho más rápidos y redes mucho más rápidas entre los nodos que se encuentran en WSCs porque las aplicaciones HPC son más interdependientes y se comunican más frecuentemente. Los diseños de HPC también tienden a utilizar hardware personalizado, especialmente en la red, de modo que a menudo no reciben el beneficio de utilizar chips de productos básicos. Por ejemplo, el IBM power 7 microprocessor solo cuesta más y usa más energía que un nodo de servidor completo en un WSC de google. Los clústeres de HPC también tienden a tener trabajos largos que mantienen a los servidores totalmente utilizados, incluso durante semanas, mientras que la utilización de servidores en WSCs oscila entre 10 % y 50 % y varía día a día.

Los operadores de un centro de datos generalmente recogen máquinas y software de terceros de muchas partes de una organización y las administran de forma centralizada para otros. Su enfoque principal tiende a ser la consolidación de los muchos servicios en menos máquinas, que están aisladas entre sí para proteger la información sensible. Por lo tanto, las máquinas virtuales son cada vez más importantes en los centros de datos. A diferencia de los WSCs, los centros de datos tienden a tener una gran cantidad de hardware y software de heterogeneidad . Los programadores de WSC personalizan el software de terceros o construyen sus propios software, y los WSC tienen hardware mucho más homogéneo; El objetivo de la WSC es hacer que el hardware / software en el almacén actúe como un solo ordenador que típicamente enfrenta una variedad de aplicaciones. A menudo el costo más grande en un centro de datos es la gente para mantenerlo, mientras que en un WSC bien diseñado el hardware del servidor es el costo más alto y el costo de la gente cambia de lo más alto a casi irrelevante.

VI. MapReduce

Además de los servicios de Internet públicos como la búsqueda, el uso compartido de video y las redes sociales que hacen famosos los WSC estos también ejecutan aplicaciones por lotes, como convertir videos en nuevos formatos o crear índices de búsqueda desde rastreos web. Hoy en día, el framework más popular para el procesamiento por lotes en un WSC es MapReduce y su gemelo de código abierto Hadoop.

MapReduce es un modelo de programación para dar soporte a la computación paralela sobre grandes colecciones de datos en grupos de computadoras. El nombre del framework está inspirado en los nombres de dos importantes métodos: Map y Reduce. MapReduce ha sido adoptado mundialmente, ya que existe una implementación OpenSource denominada Hadoop. Su desarrollo fue liderado inicialmente por Yahoo y actualmente lo realiza el proyecto Apache. Se emplea en la resolución práctica de algunos algoritmos susceptibles de ser paralelizados. No obstante MapReduce no es

[4]son colecciones de computadoras independientes que están conectadas entre sí mediante redes locales estándar (LAN) y conmutadores fuera de la plataforma

la solución para cualquier problema, de la misma forma que cualquier problema no puede ser resuelto eficientemente por MapReduce. Por regla general se abordan problemas con datasets de gran tamaño, alcanzando los petabytes de tamaño.

El Framework MapReduce tiene una arquitectura maestro / esclavo. Cuenta con un servidor maestro o JobTracker y varios servidores esclavos o TaskTrackers, uno por cada nodo.

El JobTracker es el punto de interacción entre los usuarios y el framework MapReduce. Los usuarios envían trabajos MapReduce al JobTracker, que los pone en una cola de trabajos pendientes y los ejecuta en el orden de llegada. El JobTracker gestiona la asignación de tareas y delega las tareas a los TaskTrackers. Los TaskTrackers ejecutan tareas bajo la orden del JobTracker y también manejan el movimiento de datos entre la fase Map y Reduce.

Las características del JobTracker y TaskTracker son:

- JobTracker
  - Capacidad para manejar metadatos de trabajos.
  - Decide sobre la programación.
  - Hay exactamente un JobTracker por WSC.
  - Recibe peticiones de tareas enviadas por el cliente.
  - Programa y monitoriza los trabajos MapReduce con TaskTrackers.
- TaskTracker
  - Ejecuta las solicitudes de trabajo de JobTrackers.
  - Obtiene el código que se ejecutará.
  - Aplica la configuración especifica del trabajo.
  - Comunicación con el JobTracker:
    - Envíos de la salida, finalizar tareas, actualización de tareas, etc.

Las funciones Map y Reduce están definidas ambas con respecto a datos estructurados en tuplas del tipo (clave, valor).

Map toma uno de estos pares de datos con un tipo en un dominio de datos, y devuelve una lista de pares en un dominio diferente:

$$\text{Map(k1,v1)} \rightarrow \text{list(k2,v2)}$$

La función map(): se encarga del mapeo y es aplicada en paralelo para cada ítem en la entrada de datos. Esto produce una lista de pares (k2,v2) por cada llamada. Después de eso, el framework de MapReduce junta todos los pares con la misma clave de todas las listas y los agrupa, creando un grupo por cada una de las diferentes claves generadas. Desde el punto de vista arquitectural el nodo JobTracker toma el input, lo divide en pequeñas piezas o problemas de menor identidad, y los distribuye a los denominados TaskTracker nodes. Un TaskTracker node puede volver a sub-dividir, dando lugar a una estructura arbórea. El TaskTracker node procesa el problema y pasa la respuesta al nodo JobTracker.

La función reduce es aplicada en paralelo para cada grupo, produciendo una colección de valores para cada dominio:

$$\text{Reduce(k2, list (v2))} \rightarrow \text{list(v3)}$$

La función reduce(): cada llamada a Reduce típicamente produce un valor v3 o una llamada vacía, aunque una llamada puede retornar más de un valor. El retorno de todas esas llamadas se recoge como la lista de resultado deseado.
Por lo tanto, el framework MapReduce transforma una lista de pares (clave, valor) en una lista de valores. Este comportamiento es diferente de la combinación ”map and reduce”de programación funcional, que acepta una lista arbitraria de valores y devuelve un valor único que combina todos los valores devueltos por mapa.

La función map() se ejecuta de forma distribuida a lo largo de varias máquinas. Los datos de entrada, procedentes por regla general de un gran archivo (fichero), se dividen en un conjunto de M particiones de entrada de generalmente 16 a 64 megabytes. Estas particiones pueden ser procesadas en diversas máquinas. En una invocación de MapReduce suelen ocurrir varias operaciones:

- Se procede a dividir las entradas en M particiones de tamaño aproximado de 16 a 64 megabytes. El programa MapReduce se comienza a instanciar en las diversas máquinas. Por regla general, el número de instancias se configura en las aplicaciones.
- Una de las copias del programa es especial y toma el papel de JobTracker. El resto de copias se denominan como Tasktracker y reciben la asignación de sus tareas desde el JobTracker. Se considera que existen una cantidad de M map() tareas y de R reduce(). El JobTracker se encarga de recopilar TaskTracker en reposo (es decir sin tarea asignada) y le asignará una tarea específica de map() o de reduce(). Un TaskTracker sólo puede tener tres estados: reposo, trabajando, completo.

- Un TaskTracker que tenga asignada una tarea específica de map() tomará como entrada la partición que le corresponda. Se dedicará a parsear los pares (clave, valor) para crear una nueva pareja de salida, tal y como se específica en su programación. Los pares clave y valor producidos por la función map() se almacenan como buffer en la memoria.
- Periódicamente, los pares clave-valor almacenados en el buffer se escriben en el disco local, repartidos en R regiones. Las regiones de estos pares clave-valor son pasados al JobTracker, que es responsable de redirigir a los TaskTracker que tienen tareas de reduce().
- Cuando un TaskTracker de tipo reduce es notificado por el JobTracker con la localización de una partición, éste emplea llamadas remotas para hacer lecturas de la información almacenada en los discos duros de los diversos TaskTracker de tipo map(). Cuando un TaskTracker de tipo reduce() lee todos los datos intermedios, ordena las claves de tal modo que se agrupen los datos encontrados que poseen la misma clave. El ordenamiento es necesario debido a que, por regla general, muchas claves de funciones map() diversas pueden ir a una misma función reduce(). En aquellos casos en los que la cantidad de datos intermedios sean muy grandes, se suele emplear un ordenamiento externo.
- El TaskTracker de tipo reduce() itera sobre el conjunto de valores ordenados intermedios, y lo hace por cada una de las claves únicas encontradas. Toma la clave y el conjunto de valores asociados a ella y se los pasa a la función reduce(). La salida de reduce() se añade al archivo (fichero) de salida de MapReduce.
- Cuando todas las tareas map() y reduce() se han completado, el JobTracker levanta al programa del usuario. Llegados a este punto la llamada MapReduce retorna el control al código de un usuario.

Se considera que ha habido un final de las tareas cuando este control se ha devuelto al usuario. Las salidas se ditribuyen en un fichero completo, o en su defecto se reparten en R ficheros. Estos R ficheros pueden ser la entrada de otro MapReduce o puede ser procesado por cualquier otro programa que necesite estos datos.

VI-A. Tolerancia a fallos

El mecanismo de MapReduce es tolerante a fallos cuando uno de los TaskTracker se ve sometido a un fallo. Como MapReduce se ha diseñado para procesos en los que se encuentran involucrados grandes tamaños de datos mediante el empleo de cientos o miles de ordenadores. Aún siendo la probabilidad de fallo baja, es muy posible que uno (o varios) de los TaskTracker quede desactivado precisamente por fallo de la máquina que le daba soporte. El JobTracker periódicamente hace ping a cada TaskTracker para comprobar su estatus.

Si no existe respuesta tras un cierto instante de espera, el JobTracker interpreta que el TaskTracker está desactivado. Cualquier tarea map() que ha sido completa por el TaskTracker regresa de inmediato a su estado de espera, y por lo tanto puede resultar elegible para su asignación en otros TaskTracker. De forma similar, cualquier función map() o reduce() que se encuentre en progreso durante el fallo, se resetea a estado de reposo pudiendo ser elegida para su nueva re-asignación.

Las tareas de map() completados se vuelven a re-ejecutar ante un fallo debido en parte a que su salida se almacena en los discos locales de la máquina que falló, y por lo tanto se consideran inaccesibles. Las tareas reduce() completas no son necesarias volver a ser re-ejecutadas debido a que su salida se ha almacenado en el sistema global. cuando la tarea de map() se ejecuta por un TaskTracker A y luego por un TaskTracker B (debido principalmente a un fallo), en este caso todas las tareas reduce() son notificadas para que eliminen datos procedentes del TaskTracker A y acepten las del TaskTracker B. De esta forma la ejecución de MapReduce es resiliente.

VI-B. ejemplo

ejemplo extraido de wikipedia.

Este ejemplo de MapReduce es un proceso para contar las apariciones de cada palabra en un conjunto de documentos:

```
- map(String name, String document):
-     // clave: nombre del documento
-     // valor: contenido del documento
-     for each word w in document:
-         EmitIntermediate(w, 1);
```

La función map() en este caso divide un documento en palabras (es decir lo tokeniza) mediante el empleo de un simple analizador léxico, y emite una serie de tuplas de la foma (clave, valor) donde la clave es la palabra y el valor es ”1”. Es decir, por ejemplo, del documento ”La casa de la pradera”la función map retornaría: (”la”, ”1”), (çasa”, ”1”), (”de”, ”1”), (”la”, ”1”), (”pradera”, ”1”).

```
- reduce(String word, Iterator partialCounts):
-       // word: una palabra
-      int result = 0;
-       for each v in partialCounts:
-          result += ParseInt(v);
-       Emit(result);
```

Aquí, cada documento es dividido en palabras, y cada palabra se cuenta con valor inicial ”1”por la función Map, utilizando la palabra como el resultado clave. El framework reúne todos los pares con la misma clave y se alimenta a la misma llamada Reduce, por lo tanto, esta función sólo necesita la suma de todos los valores de su entrada para encontrar el total de las apariciones de esa palabra. En el ejemplo anterior (”la”, ”1”) aparece dos veces debido a que la clave ”la”tiene dos ocurrencias, el resto de claves sólo aparece una vez.

VII. Aplicaciones en colombia

Los wsc se pueden implantar en Colombia para mejor los servicios de consulta en las elecciones, ya que es casi imposible realizar consultas acerca de los resultados de las votaciones luego de finalizado los conteos, otra implementación de los wsc seria mejorar el servicio matrículas vía web universitarias o consultas que en el caso de nuestra universidad, estos servicio se saturan cuando hay una gran cantidad de estudiantes intentan realizar sus cualquiera de estas acciones.

VIII. Conclusiones

Se puede decir que en cierto sentido los warehouse-scale computing son simples, son sólo unos pocos servidores unidos a través de una red local. En realidad, la construcción de una plataforma de computación a gran escala rentable que tiene los requisitos de fiabilidad y programación. Podemos identificar también que los WSC no son comparables con los datacenters pues tienen implementaciones diferentes.

IX. Refencias

[1] Jeff. Qué es HPC? . jsantillan[en línea] . 19 de junio 2014. [fecha de consulta: 27 de junio de 2017], disponible en: ¡https://jsantillan.wordpress.com/2008/06/19/que-es-hpc/¿.

[2] staff Wikipedia. Programación lineal. Wikipedia [en línea]. 3 de febrero de 2014. [fecha de consulta: 27 de junio de 2017], disponible en: <https://es.wikipedia.org/wiki/Programaci %C3 %B3n_lineal>.

[3] staff Wikipedia. Modelo de programación en paralelo. Wikipedia [en línea]. 15 de junio de 2017. [fecha de consulta: 27 de junio de 2017], disponible en: <https://es.wikipedia.org/wiki/Modelo_de_programaci % C3 %B3n_paralela>.

[4] staff Wikipedia. Paralelismo(informática). Wikipedia [en línea]. 12 de junio de 2017. [fecha de consulta: 27 de junio de 2017], disponible en: <https://es.wikipedia.org/wiki/Modelo_de_programaci % C3 %B3n_paralela¿.

[5] Bryan Salazar López. Método simplex. IngenieriaIndustrialEnLinea [en línea]. 12 de septiembre de 2014. [fecha de consulta: 27 de junio de 2017], disponible en: <https://www.ingenieriaindustrialonline.com/herramientas-para-el-ingeniero-industrial/investigaci %C3 %B3n-de-operaciones/m %C3 %A9todo-simplex/ >.

[6] Esteban F. Corrales. Paralelismo a nivel de instrucción [pdf]. 10 de septiembre de 2016. [fecha de consulta: 27 de junio de 2017], disponible en: <http://www.dimare.com/adolfo/cursos/2008-1/pp-InstParalel.pdf>.

[7] staff Wikipedia. Paralelismo de datos. Wikipedia [en línea]. 20 de agosto de 2010. [fecha de consulta: 27 de junio de 2017], disponible en: <https://es.wikipedia.org/wiki/Paralelismo_de_datos>.

[8] staff Wikipedia. Paralelismo de tareas. Wikipedia [en línea]. 31 de octubre de 2008. [fecha de consulta: 27 de junio de 2017], disponible en: <https://es.wikipedia.org/wiki/Paralelismo_de_tareas>.

[9] staff Wikipedia. Escalabilidad. Wikipedia [en línea]. 20 de septiembre de 2015. [fecha de consulta: 27 de junio de 2017], disponible en: <https://es.wikipedia.org/wiki/Escalabilidad>.

[10] staff Wikipedia. MapReduce. Wikipedia [en línea]. 3 enero de 2015. [fecha de consulta: 27 de junio de 2017], disponible en: <https://es.wikipedia.org/wiki/MapReduce>.

[11] hendrik wacke. ¿que es el balanceo de carga?. ComputerWorld [en línea]. 9 de enero de 2000. [fecha de consulta: 27 de junio de 2017], disponible en: <http://www.computerworld.es/tendencias/que-es-el-balanceo-de-carga>.

[12] Yolanda Olmedo. ¿que es MapReduce?. SolidQ [en línea]. 20 de agosto de 2012. [fecha de consulta: 27 de junio de 2017], disponible en: <http://blogs.solidq.com/es/big-data/que-es-mapreduce/>.