Uso de MapReduce y equilibrio de la carga en nube

Hadoop MapReduce and virtualization mejora la performance de los nodos

Kirpal A. Venkatesh
System Engineer, Global Business Services
IBM

Kishorekumar Neelamegam
Systems Engineer/IT Architect, Global Business Services
IBM

R. Revathy
Intern
IBM

03-02-2011

Conozca cómo implementar la estructura Hadoop MapReduce en un entorno en nube y cómo utilizar el equilibrio de la carga virtual para mejorar la performance tanto de un sistema de nodo único como en uno de múltiples nodos.

La computación en nube está diseñada para proveer los recursos on demand o los servicios de Internet, generalmente a escala y con el nivel de confiabilidad de un centro de datos. MapReduce es un modelo de programación diseñado para el procesamiento en paralelo de grandes volúmenes de datos dividiendo el trabajo en un grupo de tareas independientes. Es un estilo de programación en paralelo que está soportado por algunas nubes del estilo de capacity-on-demand tales como BigTable de Google Hadoop y Sector.

En este artículo, se utiliza un algoritmo de equilibrio de la carga que sigue el enfoque de la técnica Randomized Hydrodynamic Load Balancing (encontrará más detalles sobre este tema en las siguientes secciones). La virtualización es utilizada para reducir la cantidad real de servidores físicos y el costo; más importante aún, la virtualización es usada para lograr el uso eficiente de la CPU de una máquina física.

Para aprovechar al máximo este artículo, debería tener una idea general de los conceptos sobre la computación en nube, la técnica Randomized Hydrodynamic Load Balancing y el modelo de programación Hadoop MapReduce. Una comprensión básica de la programación en paralelo

ayudará y todo conocimiento sobre Java™ u otro lenguaje orientado a los objetos será una buena herramienta de soporte.

Para este artículo, el algoritmo MapReduce fue implementado en un sistema utilizando:

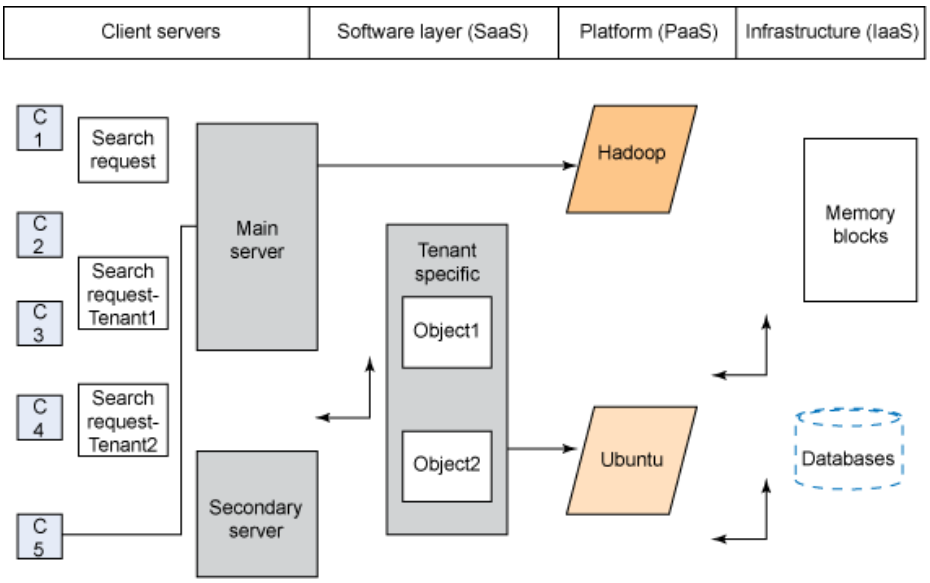
- Hadoop 0.20.1.
- Eclipse IDE 3.0 o superior (o Rational Application Developer 7.1).
- Ubuntu 8.2 o superior.

Antes de profundizar en el algoritmo MapReduce estableceremos los conceptos básicos suficientes, por lo menos, para el presente artículo, sobre la arquitectura en nube, el equilibrio de la carga, MapReduce y la programación en paralelo —.

Arquitectura en nube: Conceptos básicos

La Figura 1 muestra un cuadro detallado del sistema completo, las plataformas, el software y cómo son utilizados para lograr el objetivo establecido en el presente artículo.

Figura 1. Arquitectura en nube



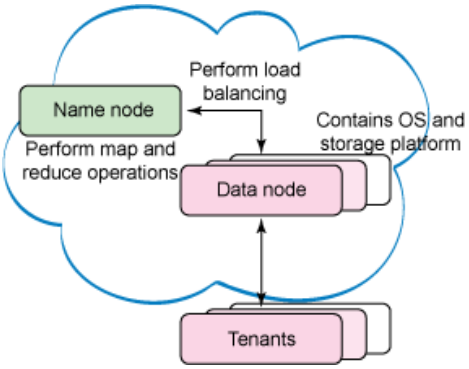
Puede ver que Ubuntu 9.04 y 8.2 son usados para los sistemas operativos; Hadoop 0.20.1, Eclipse 3.3.1 y Sun Java 6, para las plataformas; el lenguaje Java, para la programación, y HTML, JSP y XML, como lenguajes de scripting.

Esta arquitectura en nube tiene ambos nodos, maestro y esclavo. En esta implementación, un servidor principal es mantenido, el que obtiene las solicitudes del cliente y las maneja dependiendo del tipo de solicitud. El nodo maestro está presente en el servidor principal y los nodos esclavos, en el servidor secundario.

Las solicitudes de búsqueda son enviadas al NameNode de Hadoop, presente en el servidor principal como puede ver en la Figura 2. Entonces el Hadoop NameNode se ocupa de la operación de búsqueda e indexación iniciando una gran cantidad de procesos Map y Reduce. Una

vez que la operación MapReduce para una clave de búsqueda determinada es completada, el NameNode le devuelve el valor de la salida al servidor y a su vez al cliente.

Figura 2. Las funciones Map y Reduce realizan la búsqueda y la indexación



Si la solicitud es para un software determinado, los pasos para la autenticación son llevados a cabo basándose en la ID de tenant del cliente, los pagos debidos, la elegibilidad para usar un software determinado, y el período de lease del software. Entonces el servidor le presta servicio a esta solicitud y le permite al usuario consumir una combinación seleccionada de software.

El dispositivo de multitenancy de SaaS se provee acá, en el que un único caso del software sirve a una cantidad de tenants. Para cada solicitud específico de tenant habrá una línea delgada de aislamiento generada basada en la id de tenant. Estas solicitudes son atendidas por un único caso.

Cuando una solicitud de cliente específico de tenant desea buscar archivos o consumir cualquier otro software de multi-tenant, los ofrecimientos usan Hadoop en el caso del sistema operativo provisto (PaaS). También, a fin de guardar sus datos -- tal vez una base de datos o archivos-- en la nube, el cliente tendrá que tomar algo del espacio de la memoria del centro de datos (IaaS). Todos estos movimientos son transparentes para el usuario final.

Randomized Hydrodynamic Load Balancing: Conceptos básicos

El equilibrio de la carga es utilizado para asegurarse de que ninguno de sus recursos existentes esté ociosos mientras que otros están siendo utilizados. Para equilibrar la distribución de la carga, usted puede migrar la carga desde los *nodos fuente* (que tienen una carga de trabajo excedente) hasta los *nodos destino* que en comparación están levemente cargados.

Cuando usted aplica el equilibrio de carga durante el tiempo de ejecución, se denomina *equilibrio dinámico de carga*— esto puede ser realizado tanto en una manera directa como en una manera iterativa de acuerdo con la selección del nodo de ejecución:

- En los métodos iterativos, el nodo de destino final es determinado a través de varios pasos de interacción.
- En los métodos directos, el nodo de destino final es seleccionado en un paso.

Para este artículo, se utiliza el método Randomized Hydrodynamic Load Balancing, un método híbrido que aprovecha tanto el método directo como el iterativo.

MapReduce: Conceptos básicos

Los programas MapReduce están diseñados para computar grandes volúmenes de datos en paralelo. Esto requiere abocarse a la carga de trabajo en una gran cantidad de máquinas. Hadoop provee una forma sistemática de implementar este paradigma de programación.

La computación toma un grupo de pares de claves/valores de entrada y produce un grupo de pares de claves/valores de salida. La computación involucra dos operaciones básicas: Map y Reduce.

La operación Map, escrita por el usuario, toma un par de entrada y produce un grupo intermedio de pares de claves/valores. La bibliteca de MapReduce agrupa todos los valores intermedios asociados con la misma Clave #1 intermedia y los pasa a la función Reduce.

La función Reduce, también escrita por el usuario, acepta una Clave #1 intermedia y un grupo de valores para esa clave. Fusione dichos valores para formar un grupo posiblemente más pequeño de valores. Normalmente sólo un valor de salida de 0 o 1 es producido por invocación de Reduce. Los valores intermedios les son suministrados a la función Reduce del usuario a través de un iterator (un objeto que le permite a un programador atravesar todos los elementos de un conjunto independientemente o de su implementación específica). Esto le permite a usted manejar las listas de valores que sean demasiado grandes para la memoria.

Tome el ejemplo del problema de WordCount. Ésta contará la cantidad de ocurrencias de cada palabra en un gran conjunto de documentos. Las funciones Mapper y Reducer aparecerán como en el Listado 1.

Listado 1. Map y Reduce en un problema de WordCount

```
mapper (filename, file-contents):
  for each word in file-contents:
    emit (word, 1)

reducer (word, values):
  sum = 0
  for each value in values:
    sum = sum + value
  emit (word, sum)
```

La función Map emite cada palabra además de una cuenta de ocurrencias asociada. La función Reduce suma todas las cuentas emitidas para una palabra determinada. Esta funcionalidad básica, al ser creada en los clusters, puede convertirse fácilmente en un sistema de procesamiento en paralelo de alta velocidad.

La computación en grandes volúmenes de datos ha sido realizada antes, generalmente en una configuración distribuida. Lo que hace que Hadoop sea único es su modelo simplificado de programación — lo que le permite al usuario escribir y probar rápidamente los sistemas distribuidos —, y su eficiente distribución automática de datos y trabajo en las máquinas, utilizando de una en una el paralelismo subyacente de los cores de la CPU.

Tratemos de aclarar las cosas un poco más. Como se habló anteriormente, en un cluster de Hadoop usted tiene los siguientes nodos:

- El NameNode (el maestro en nube).
- Los DataNodes (o los esclavos).

Los nodos en el cluster han cargado previamente los archivos locales de entrada. Cuando se inicia el proceso de MapReduce, el NameNode usa el proceso `JobTracker` para asignar las tareas que serán realizadas por DataNodes, a través de los procesos de `TaskTracker`. Varios procesos Map se ejecutan en cada DataNode y se le darán los resultados intermedios al proceso de fusión que genera, por ejemplo, el conteo de palabras en el archivo de una máquina (como en el caso de un problema de WordCount). Los valores son movidos y enviados a los procesos Reduce que generan la salida para el problema en cuestión.

De qué manera se utiliza el equilibrio de la carga

El equilibrio de la carga es útil al distribuir la misma en una manera equitativa en los nodos libres cuando un nodo es cargado por encima del nivel de su umbral. Aunque el equilibrio de la carga no es importante en la ejecución de un algoritmo MapReduce se vuelve esencial al manejar archivos grandes para el procesamiento y cuando el uso de los recursos de hardware crítico. Como aspecto a destacar mejora la utilización del hardware en las situaciones críticas en cuanto a los recursos con una ligera mejora en la performance.

Un módulo fue implementado para equilibrar el uso del espacio en disco en un cluster de Hadoop Distributed File System cuando algunos nodos de datos se completaron o cuando los nuevos nodos vacíos se unieron al cluster. El balanceador (herramienta Class Balancer) fue iniciado con un valor de umbral; este parámetro es una fracción entre 0 y 100 por ciento con un valor por omisión de 10 por ciento. Esto define el destino para por si el cluster es equilibrado; cuanto más pequeño sea el valor del umbral, más equilibrado será el cluster. También lleva más tiempo ejecutar el balanceador. (Nota: Un valor de umbral puede ser tan pequeño que usted no pueda equilibrar el estado del cluster porque las aplicaciones pueden estar escribiendo y eliminando los archivos al mismo tiempo).

Se considera que un cluster está equilibrado si para cada nodo de datos la proporción del espacio utilizado en el nodo con respecto a la capacidad total del mismo (conocido como la *utilización del nodo*) difiere del promedio del espacio utilizado en el cluster con respecto a la capacidad total del cluster (*utilización del cluster*) en no más que el valor del umbral.

El módulo mueve bloques desde los nodos de datos que están siendo muy utilizados a los que son poco utilizados en una manera iterativa; en cada iteración un nodo mueve o recibe no más que la fracción del umbral de su capacidad y cada iteración funciona no más de 20 minutos.

En esta implementación, los nodos son clasificados como *altamente utilizados*, *medianamente utilizados* y *poco utilizados*. Dependiendo de la clasificación de la utilización de cada nodo, la carga fue transferida entre los nodos y el cluster fu equilibrado. El módulo funcionó de la siguiente manera:

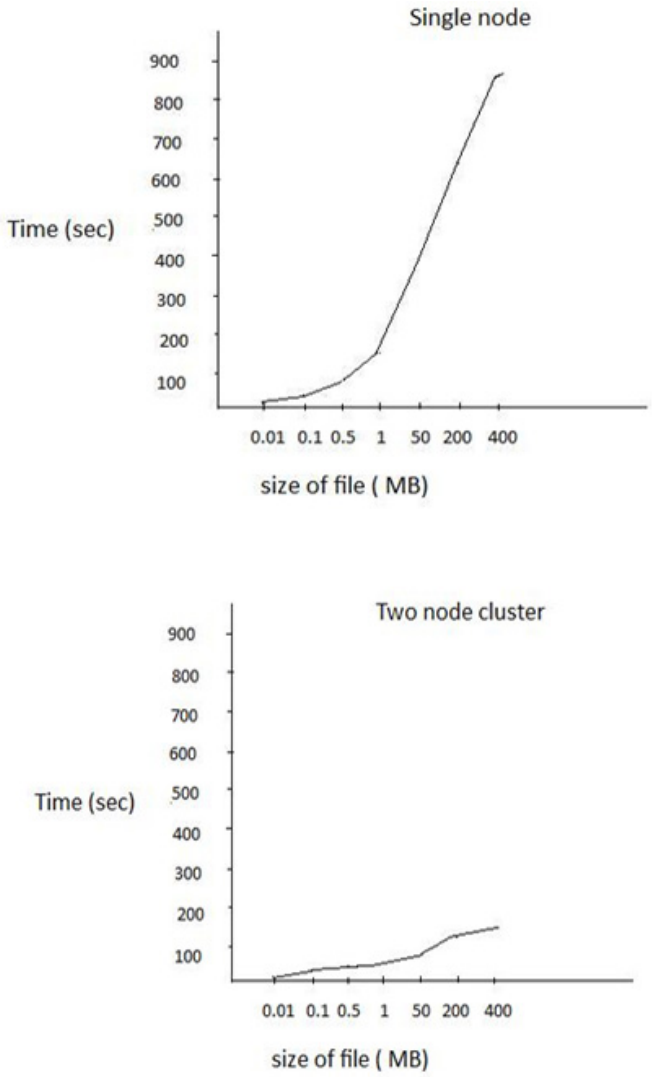
- Primero, adquiere los detalles del vecindario:
  1. Cuando la carga aumenta en un DataNode al nivel del umbral le envía una solicitud al NameNode.
  2. El NameNode tenía información acerca de los niveles de carga de los vecinos específicos más cercanos de DataNode.
  3. Las cargas son comparadas por el NameNode y luego los detalles acerca de los nodos vecinos free-est son enviados al DataNode específico.
- Luego, los DataNodes van a funcionar:
  1. Cada DataNode compara la cantidad de su propia carga con la suma de la cantidad de carga de su vecinos más cercanos.
  2. Si el nivel de carga de un DataNode es mayor que la suma de sus vecinos, entonces los nodos de destino de la carga (vecinos directos Y otros nodos) serán elegidos al azar.
  3. Luego las solicitudes de carga son enviadas a los nodos destino.
- Por último, la solicitud es recibida:
  1. Los buffers son mantenidos en cada nodo para recibir las solicitudes de carga.
  2. Una message passing interface (MPI) administra este buffer.
  3. Un thread principal atenderá la cola en buffer y les brindará servicio a las solicitudes que reciba.
  4. Los nodos entran en la fase de carga-equilibrio-ejecución.

Evaluación de la performance

Fueron provistos diferentes grupos de archivos de entrada, cada uno de distinto tamaño y fueron ejecutadas las tareas de MapReduce tanto en el cluster de nodo único como en el de dos nodos. Los correspondientes tiempos de ejecución fueron medidos y llegamos a la conclusión de que ejecutar MapReduce en clusters es mucho más eficiente para un gran volumen de archivos de entrada.

Los gráficos en la Figura 3 ilustran los resultados de la performance de ejecutar varios nodos.

Figura 3. El equilibrio de carga de MapReduce funciona más eficientemente en clusters



En conclusión

Nuestro experimento con Hadoop MapReduce y el equilibrio de carga conduce a dos conclusiones ineludibles:

- En un entorno en nube, la estructura de MapReduce incrementa la eficiencia en el rendimiento para los grandes volúmenes de datos. En contraste, usted no vería necesariamente dicho incremento en el rendimiento en un sistema que no sea en nube.
- Cuando el conjunto de datos es pequeño, MapReduce y el equilibrio de carga no logra un apreciable aumento en el rendimiento en un sistema en nube.

Por lo tanto, considere una combinación del procesamiento en paralelo del estilo de MapReduce y el equilibrio de carga al planificar el proceso de una gran cantidad de datos en su sistema en nube.

Sobre los autores

Kirpal A. Venkatesh

Kirpal A. Venkatesh es Ingeniero de Sistemas en Global Business Service en IBM, profesional de las tecnologías Microsoft, seguidor e innovador entusiasta de la tecnología de la computación en nube.

Kishorekumar Neelamegam

Kishorekumar es Arquitecto de IT de Global Business Service en IBM. Tiene más de 13 años de experiencia en el desarrollo de software con una gran atención en la integración de software a la plataforma de Rational. Apasionado evangelista en nube, Kishore es un frecuente participante en developerWorks. Usted puede seguir sus actividades a través de su MydW profile y su blog.

R. Revathy

R. Revathy es estudiante del último año en el College of Engineering, Guindy, Anna University, Chennai, India.