Using MapReduce and load balancing on the cloud

Hadoop MapReduce and virtualization improves node performance

Kirpal A. Venkatesh  
Kishorekumar Neelamegam  
R. Revathy  

July 19, 2010

Learn how to implement the Hadoop MapReduce framework in a cloud environment and how to use virtual load balancing to improve the performance of both a single- and multiple-node system.

Cloud computing is designed to provide on demand resources or services over the Internet, usually at the scale and with the reliability level of a data center. MapReduce is a programming model designed for processing large volumes of data in parallel by dividing the work into a set of independent tasks. It is a style of parallel programming that is supported by some capacity-on-demand-style clouds such as Google's BigTable, Hadoop, and Sector.

In this article, a load-balancing algorithm that follows the approach of the Randomized Hydrodynamic Load Balancing technique (more on that in the following sections) is used. Virtualization is used to reduce the actual number of physical servers and cost; more importantly, virtualization is used to achieve efficient CPU utilization of a physical machine.

To get the most from this article, you should have a general idea of cloud computing concepts, the Randomized Hydrodynamic Load Balancing technique, and the Hadoop MapReduce programming model. A basic understanding of parallel programming will help and any programming knowledge on Java™ or other object-oriented languages will be a good support tool.

For this article, the MapReduce algorithm was implemented on a system using:

- Hadoop 0.20.1.
- Eclipse IDE 3.0 or above (or Rational Application Developer 7.1).
- Ubuntu 8.2 or above.

Before diving into the MapReduce algorithm, we'll set up the basics of the cloud architecture, load balancing, MapReduce, and parallel programming — enough at least for this article.
Cloud architecture: The basics

Figure 1 shows a detailed picture of the complete system, platforms, software, and how they are used to achieve the goal set in this article.

**Figure 1. The cloud architecture**

<table>
<thead>
<tr>
<th>Client servers</th>
<th>Software layer (SaaS)</th>
<th>Platform (PaaS)</th>
<th>Infrastructure (IaaS)</th>
</tr>
</thead>
</table>

You can see Ubuntu 9.04 and 8.2 is used for the operating systems; Hadoop 0.20.1, Eclipse 3.3.1, and Sun Java 6 for the platforms; the Java language for programming; and HTML, JSP, and XML as the scripting languages.

This cloud architecture has both master and slave nodes. In this implementation, a main server is maintained that gets client requests and handles them depending on the type of request. The master node is present in main server and the slave nodes in secondary server.

Search requests are forwarded to the NameNode of Hadoop, present in main server as you can see in Figure 2. The Hadoop NameNode then takes care of the searching and indexing operation by initiating a large number of Map and Reduce processes. Once the MapReduce operation for a particular search key is completed, the NameNode returns the output value to the server and in turn to the client.

**Figure 2. Map and Reduce functions do searching and indexing**
If the request is for a particular software, authentication steps are done based on the customer tenant ID, payment dues, eligibility to use a particular software, and the lease period for the software. The server then serves this request and allows the user to consume a selected software combination.

The multitenancy feature of SaaS is provided here, in which a single instance of the software serves a number of tenants. For every tenant specific request there will be a thin line of isolation generated based on the tenant id. These requests are served by a single instance.

When a tenant specific client request wants to search files or consume any other multi-tenant software the offerings use Hadoop on the provisioned operating system instance (PaaS). Also, in order to store his data -- perhaps a database or files-- in the cloud, the client will have to take some memory space from the data center (IaaS). All these moves are transparent to the end user.

**Randomized Hydrodynamic Load Balancing: The basics**

Load balancing is used to make sure that none of your existing resources are idle while others are being utilized. To balance load distribution, you can migrate the load from the source nodes (which have surplus workload) to the comparatively lightly loaded destination nodes.

When you apply load balancing during runtime, it is called dynamic load balancing— this can be realized both in a direct or iterative manner according to the execution node selection:

- In the iterative methods, the final destination node is determined through several iteration steps.
- In the direct methods, the final destination node is selected in one step.

For this article, the Randomized Hydrodynamic Load Balancing method is used, a hybrid method that takes advantage of both direct and iterative methods.

**MapReduce: The basics**

MapReduce programs are designed to compute large volumes of data in a parallel fashion. This requires dividing the workload across a large number of machines. Hadoop provides a systematic way to implement this programming paradigm.

The computation takes a set of input key/value pairs and produces a set of output key/value pairs. The computation involves two basic operations: Map and Reduce.

The Map operation, written by the user, takes an input pair and produces a set of intermediate key/value pairs. The MapReduce library groups together all intermediate values associated with the same intermediate Key #1 and passes them to the Reduce function.

The Reduce function, also written by the user, accepts an intermediate Key #1 and a set of values for that key. It merges together these values to form a possibly smaller set of values. Typically just an output value of 0 or 1 is produced per Reduce invocation. The intermediate values are supplied to the user's Reduce function via an iterator (an object that allows a programmer to traverse
through all the elements of a collection regardless of its specific implementation). This allows you to handle lists of values that are too large to fit in memory.

Take the example of WordCount problem. This will count the number of occurrences of each word in a large collection of documents. The Mapper and Reducer function will look like Listing 1.

### Listing 1. Map and Reduce in a WordCount problem

```java
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)
```

The Map function emits each word plus an associated count of occurrences. The Reduce function sums together all counts emitted for a particular word. This basic functionality, when built over clusters, can easily turn into a high-speed parallel processing system.

Performing computation on large volumes of data has been done before, usually in a distributed setting. What makes Hadoop unique is its simplified programming model — which allows the user to quickly write and test distributed systems — and its efficient, automatic distribution of data and work across machines and in turn utilizing the underlying parallelism of the CPU cores.

Let’s try to make things a little clearer. As discussed earlier, in a Hadoop cluster you have the following nodes:

- The NameNode (the cloud master).
- The DataNodes (or the slaves).

Nodes in the cluster have preloaded local input files. When the MapReduce process is started, the NameNode uses the JobTracker process to assign tasks which have to be carried out by DataNodes, through the TaskTracker processes. Several Map processes will run in each DataNode and the intermediate results will be given to the combiner process which generates, for instance, the count of words in file of one machine as (in case of a WordCount problem). Values are shuffled and sent to Reduce processes which generate the final output for the problem of interest.

### How load balancing is used

Load balancing is helpful in spreading the load equally across the free nodes when a node is loaded above its threshold level. Though load balancing is not so significant in execution of a MapReduce algorithm, it becomes essential when handling large files for processing and when hardware resources use is critical. As a highlight, it enhances hardware utilization in resource-critical situations with a slight improvement in performance.

A module was implemented to balance the disk space usage on a Hadoop Distributed File System cluster when some data nodes became full or when new empty nodes joined the cluster. The
balancer (Class Balancer tool) was started with a threshold value; this parameter is a fraction between 0 and 100 percent with a default value of 10 percent. This sets the target for whether the cluster is balanced; the smaller the threshold value, the more balanced a cluster will be. Also, the longer it takes to run the balancer. (Note: A threshold value can be so small that you cannot balance the state of the cluster because applications may be writing and deleting files concurrently.)

A cluster is considered balanced if for each data node, the ratio of used space at the node to the total capacity of node (known as the utilization of the node) differs from the the ratio of used space at the cluster to the total capacity of the cluster (utilization of the cluster) by no more than the threshold value.

The module moves blocks from the data nodes that are being utilized a lot to the poorly used ones in an iterative fashion; in each iteration a node moves or receives no more than the threshold fraction of its capacity and each iteration runs no more than 20 minutes.

In this implementation, nodes are classified as highly-utilized, average-utilized, and under-utilized. Depending upon the utilization rating of each node, load was transferred between nodes and the cluster was balanced. The module worked like this:

- First, it acquires neighborhood details:
  1. When the load increases in a DataNode to the threshold level, it sends a request to the NameNode.
  2. The NameNode had information about the load levels of the specific DataNode's nearest neighbors.
  3. Loads are compared by the NameNode and then the details about the free-est neighbor nodes are sent to the specific DataNode.
- Next, the DataNodes go to work:
  1. Each DataNode compares its own load amount with the sum of the load amount of its nearest neighbors.
  2. If a DataNode's load level is greater than the sum of its neighbors, then load-destination nodes (direct neighbors AND other nodes) will be chosen at random.
  3. Load requests are then sent to the destination nodes.
- Last, the request is received:
  1. Buffers are maintained at every node to received load requests.
  2. A message passing interface (MPI) manages this buffer.
  3. A main thread will listen to the buffered queue and will service the requests it receives.
  4. The nodes enter the load-balancing-execution phase.

**Evaluating the performance**

Different sets of input files were provided, each of different size, and executed the MapReduce tasks in both single- and two-node clusters. The corresponding times of execution were measured and we came to the conclusion that running MapReduce in clusters is by far the more efficient for a large volume of input file.

The graphs in Figure 3 illustrate our performance results from running on various nodes.
In conclusion

Our experiment with Hadoop MapReduce and load balancing lead to two inescapable conclusions:

- In a cloud environment, the MapReduce structure increases the efficiency of throughput for large data sets. In contrast, you wouldn't necessarily see such an increase in throughput in a non-cloud system.
- When the data set is small, MapReduce and load balancing do not effect an appreciable increase in throughput in a cloud system.

Therefore, consider a combination of MapReduce-style parallel processing and load balancing when planning to process a large amount of data on your cloud system.
Related topics

- See how Map and Reduce are carried out on Hadoop.
- Find free courses on Hadoop fundamentals, stream computing, text analytics, and more at Big Data University.
- Download IBM InfoSphere BigInsights Basic Edition at no charge and build a solution that turns large, complex volumes of data into insight by combining Apache Hadoop with unique technologies and capabilities from IBM.
- While you're at it, you might want to take a look at Cloud MapReduce, another implementation of the MapReduce programming model that offers a broader architectural perspective than the original model.
- You can acquire Hadoop 0.20.1 from Apache.org.
- Get Eclipse IDE (version 3.0 or greater) from Eclipse.org.
- Get Ubuntu from Ubuntu.com.
- The developerWorks cloud computing offers updated resources on cloud computing, including An introduction to the world of cloud computing.

© Copyright IBM Corporation 2010
Trademarks
(www.ibm.com/developerworks/ibm/trademarks/)