Hadoop MapReduce Introduction

Preface

As the rapid development of hardware and software, parallel computing technique and application trend to be more and more flourished and complicated. According to different application scenarios , hardware limits and cost considerations, the distributed computing architectures could be various.

Developers can choose openMP for a small scale but computing-intensive problem, or use MPI, even hybrid system with openMP and MPI, when the single machine capability is not sufficient. However, it seems one would need more choices when the application comes to an extremely large scale. Hadoop MapReduce is a software framework which processes vast amounts of data (multi-terabyte data-sets) in-parallel on large clusters (thousands of nodes) of commodity hardware.

Unlike openMP and MPI, Hadoop doesn't use any shared memory between tasks. Hadoop by itself only allows a very restricted programming model that severely limits communication between tasks in order to allow extreme scalability of program execution. Hadoop is all about data on disk in a distributed file store whereas openMP is all about data in memory. Hadoop takes care of scheduling tasks, monitoring them and re-executes the failed tasks, which makes it more reliable, fault-tolerant to user applications.

Installation and Configuration

User could download Hadoop latest release binaries(source code) from the apache official page:

http://www.apache.org/dyn/closer.cgi/hadoop/common/

Hadoop supports running cluster in three modes: Standalone, Pseudo-Distributed and Fully Distributed. For single node, if you only want to experience the programming model or debug, you can choose Standalone mode, which is relatively easy to setup:

http://hadoop.apache.org/docs/r2.7.1/hadoop-project-dist/hadoop-common/SingleCluster.html#Standalone_Operation

Hadoop is strongly biased for Java, so make sure you have jdk installed and configured on your host properly before you try your first hello world program.

If you want to view Hadoop status and browse Hadoop filesystem, you could try Pseudo-Distributed mode:

http://hadoop.apache.org/docs/r2.7.1/hadoop-project-dist/hadoop-common/SingleCluster.html#Pseudo-Distributed_Operation

MapReduce Programming Model

Initially, users must put datasets into HDFS, which will be separately stored and managed by Hadoop filesystem. Through a mapper template interface, developer must split input datasets of program into chunks, and then define the operations performed on each chuck of maps, Hadoop will handle these map tasks in parallel manner and sort the output of map tasks (to improve performance).

The results then will be passed as the input to reduce task, which is also defined by developer via a reducer template class, thus the data of distributed maps will be reduced and combined into a relatively small scale. Users can optionally specify a combiner, to perform local aggregation of the intermediate outputs, which helps to cut down the amount of data transferred from the Mapper to the Reducer:

(input) <k1, v1> -> map -> <k2, v2> -> combine -> <k2, v2> -> reduce -> <k3, v3> (output)

The input and output of MapReduce jobs are exclusively key-value pairs, the map task splits the input dataset as key-value pairs and output them as another set of key-value pairs. The reduce task and combiner also perform aggregation to those pairs via shared key.

I follow the WordCount example in the MapReduce official site as the practice to be familiar with Hadoop features.

The input dataset is a text file or a group of text files, the output is the statistics of each unique word appeared in the files.

For the input part, I put a group of text files to Hadoop filesystem via below hdfs shell commands:

```
hadoop fs -mkdir -p /user/cchen/input
hadoop fs -put ./files/* /user/cchen/input
```

The files in '/user/cchen/input' directory will be the input datasets.

```
public static void main(String[] args) throws Exception {
   ....
   FileInputFormat.addInputPath(job, new Path(args[0]));  // input the /user/cchen/input path
   ....
 }
```

In WordCount.java, it defines a mapper to split input data:

```
public static class Map extends MapReduceBase implements Mapper<LongWritable, Text, Text,
IntWritable> {
      private final static IntWritable one = new IntWritable(1);
      private Text word = new Text();

      public void map(LongWritable key, Text value, OutputCollector<Text, IntWritable> output,
Reporter reporter) throws IOException {
         String line = value.toString();
         StringTokenizer tokenizer = new StringTokenizer(line);
```

```
            while (tokenizer.hasMoreTokens()) {
                word.set(tokenizer.nextToken());
                output.collect(word, one);
            }
        }
    }
```

This mapper splits all input files' content into lines, and pass them as the input key-value pairs into map task, each line will be treated as an individual input to perform words statistic, then map task will do the local aggregation and output the result key-value pairs set to the reducer:

```
public static class Reduce extends MapReduceBase implements Reducer<Text, IntWritable,
Text, IntWritable> {
    public void reduce(Text key, Iterator<IntWritable> values, OutputCollector<Text,
IntWritable> output, Reporter reporter) throws IOException {
        int sum = 0;
        while (values.hasNext()) {
            sum += values.next().get();
        }
        output.collect(key, new IntWritable(sum));
    }
}
```

The reducer defined here is simply to perform an aggregation on shared-key pairs, which is sum up statistics for an unique word in all map output. Thus, the final output key-value pairs of the reducer will be the overall result.

The result is correct. But to measure the execution time of the program, you have to put the time-measurement in the **close() callback of the reducer,** which is the real endpoint of computing, or else you will get a confusing performance measurement result.

A more convenient way to trace the performance status is utilizing the YARN job tracker service, refer the setup instructions:
http://hadoop.apache.org/docs/r2.7.1/hadoop-project-dist/hadoop-common/SingleCluster.html#YARN_on_a_Single_Node

Utilizing YARN, I measured the time consuming in two different cases to verify the advantage of MapReduce. By default, Hadoop assigns one core to each map process and one map process for each input file. Initially, I input only one file, then five files in input. The result show as below:

Number of files	Map Processes(cores)	Reduce Processes(cores)	Duration(sec.)
1	2	1	16
5	5	1	24

We can see from the test sequence that the time consuming is not linearly influenced by the increasing datasets, which means the advantage of MapReduce model might be promising when handling large scale dataset.

PiEstimator Program and the Performance

To compare performance of MapReduce with normal sequential code, I tried the Pi estimator program in MapReduce model and then rewrite the program with sequential flow. With Monte Carlo algorithm, I use two arguments to define the computing: samples count and maps count. Samples count defines the count of sample points for each Pi calculation, while maps count means repeat how many times of the Pi calculation. In MapReduce model, maps count also determines the map processes count.

I tried two groups parameter for the experiment, as previously mentioned, for MapReduce, by default, maps count also means processes(core) count, but for Pseudo-Distributed mode, the cluster only have 8 virtual cores (The actual host only has two cores), so the processes scheduling is far more complicated:

Model	Samples Count (Iterations of each map)	Maps Count(Map Processes)	Duration (sec.)
MapReduce	16	1000000	48
Sequential	16	1000000	8
MapReduce	1000	4000000	1800
Sequential	1000	4000000	21

The result seems confusing, the performance of sequential program overwhelm MapReduce model. It's partially caused by the overhead of MapReduce framework (jobs management, failure management, etc), but most importantly, the performance of MapReduce model is depressed by heavy IO loads.

Hadoop MapReduce model serializes the output of each mapper to HDFS storage, so more distributed map tasks led to huge amount of HDFS files reading/writing cost, which dramatically slow down the whole performance.

But that doesn't means sequential model is better than MapReduce. Because this is only a relatively small scale dataset and application scenario, in which the beneficial of MapReduce was severely depressed comparing to huge I/O and scheduling cost.

However, if you want to do aggregations on hundreds of TB of unstructured data where there is very little CPU to be spent on each input record, then Hadoop would have uncontested supremacy over sequential model, even openMP/MPI architecture.

Conclusion

Actually, it's impossible to assess the "better" architecture among various parallel computing systems. They are so different that there is going to be very little comparison between them for any given application.

For example, OpenMP is based on shared memory and efficient multi-threading. So it is good for building up programs that make very good use of a single large machine. And it is possible to build a hybrid system using MPI to pass messages between multi-threaded applications on different computers. In addition, openMP run on a single node so startup times are likely to be in the millisecond range. Even with MPI, most HPC schedulers can start programs in a few hundred milliseconds.

But for Hadoop, it doesn't use any shared memory between tasks. Hadoop only allows a very restricted programming model that severely limits communication between tasks in order to allow extreme scalability of program execution. In terms of efficiency, it is likely that using Hadoop will impose as much as an order of magnitude decrease in efficiency. In compensation, you can execute your program on a cluster with thousands of times more resources and your program will be highly tolerant of machine, disk or networking failures. Whereas openMP is all about data in memory, Hadoop is all about data on disk in a distributed file store.

In short, to achieve high performance, developers have to comprehensively consider different aspects of their application scenario and hardware limits before deploy a specific architecture.

Reference

http://hadoop.apache.org/docs/r2.7.1/index.html