AN ANALYTICAL PERFORMANCE MODEL OF MAPREDUCE

Xiao Yang, Jianling Sun

Department of Computer Science and Technology,
Zhejiang University, Hangzhou, China
yang_xiao@zju.edu.cn, sunjl@zju.edu.cn

Abstract

MapReduce is a distributed computing framework. Its application in distributed systems is a rapidly emerging field. Although this framework can leverage clusters to improve computing performance, tuning it is still challenging. Most current works related to MapReduce performance are based on system monitoring and simulation, and lack analytical performance models. In this paper, we propose a simple and general MapReduce performance model for better understanding the impact of each component on overall program performance, and verify it in a small cluster. The results indicate that our model can predict the performance of MapReduce system and its relation to the configuration. According to our model, performance can be improved significantly by modifying Map split granularity and number of reducers without modifying the framework. The model also points out potential bottlenecks of the framework and future improvement for better performance.

Keywords: performance model; MapReduce; distributed computing

1 Introduction

Enterprises, especially internet companies have high demand for large scale data processing. Parallel computing is proposed as a solution to this problem, but traditional parallel programming paradigm which includes message-passing and shared-memory threads are too cumbersome for most developers. After it was introduced by Google in 2004, MapReduce[1] rapidly gained popularity in both industry and research for its simplicity. This framework provides an efficient runtime system that handles low-level mapping, resource management, and fault tolerance issues automatically regardless of the system characteristics or scale. In 2006, Hadoop[2], an open source implementation of MapReduce, was launched by the Apache Community, making parallel computing easy to access.

Many companies are considering using Hadoop and distributed clusters for data processing. Cloud computing is an option, but some may not trust a third-party company to hold their private data, and prefer to setting their own clusters which are relatively small but can multiply the performance of some computations. However, risks exist because it is difficult to know the overall performance before deployment in production environment. Resources may be wasted if the performance improvement achieved by the cluster cannot meet the final goal. Even in production environments, performance tuning is a tedious task. Most current works related to MapReduce performance are based on system monitoring and simulation, and lack analytical performance models [13].

In this paper, we propose a simple and general MapReduce performance model for better understanding the impact of each component on overall program performance. The results indicate that our model can predict the performance of MapReduce system and its relation to the configuration. According to our model, the performance can be improved significantly by optimizing Map split granularity and number of reducers without modifying the framework itself. The model also points out potential bottlenecks and future improvement of the framework for better performance.

1.1 Related work

Many research works about performance has been done for MapReduce framework after its emergence. C. Ranger et al. [3] use 8 benchmarks to evaluate MapReduce performance. D. Jiang et al. [10] identify several factors affecting the system performance, including I/O mode, record parsing and scheduling strategy. Several log collection and analysis tools [4][6] have been developed to display MapReduce-specific behavior. 10-months of MapReduce logs from Yahoo!'s M45 supercomputing cluster are analyzed by S. Kavulya et al. [5] However, they cannot discover the root-cause of performance problems, but aids in the process through useful visualizations and analysis that users can exploit. Wang et al. [7] build a simulator for MapReduce workloads which predicts expected application performance for various MapReduce configurations. Their approach achieves high prediction accuracy, ranging between 3.42% and 19.32%. J. Berlinska et al. [12] propose a mathematical model of MapReduce, and analyze MapReduce distributed computations as a divisible load scheduling problem, but they do not consider the system constraints. There is also some research work on optimizing MapReduce [8][9][11].

The remainder of this paper is organized as follows. Section 2 introduces the process of MapReduce, and exploits all the resources it uses for each stage. Section 3 presents our performance model. In Section 4, we use a simple experiment to verify our performance model. Finally, we conclude the analysis results and point out potential improvement in Section 5.

2 MapReduce overview

This section introduces the basic concepts of MapReduce and the execution process.

2.1 Programming model

A MapReduce program is presented as Map and Reduce functions. The framework splits input data into partitions, which can be processed by user-defined Map functions, and produces <key, value> pairs as intermediate output. The Reduce functions merge all intermediate pairs with the same key, sort them and output the final <key, value> pairs. Both the map function and reduce function can be executed in parallel on non-overlapping input and intermediate data.

2.2 Execution process

In the framework, a job is submitted to a MapReduce system and distributed into task executors. The program starts after distribution. The execution process is divided into 2 primary steps: map and reduce. Each map step contains the following parts: data read, split, map function, and shuffle; each reduce step contains data transfer, sort, reduce function and output. See Fig. 1 for the execution process.

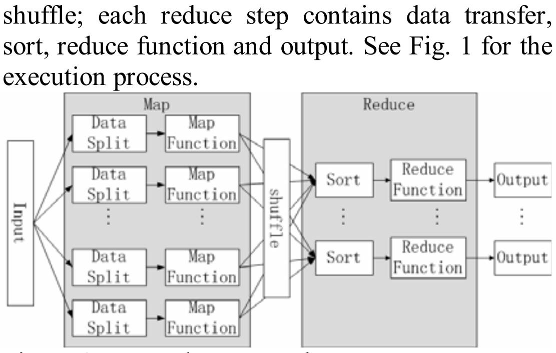

Figure 1 MapReduce execution process

3 Performance model

Our model focuses on the relation between the overall performance and the number of mappers and reducers. By analyzing the model, we can get the optimal number of mappers and reducers with limited resources. J. Berlinska's model also discusses the optimal data distribution, but it does not take into account the system limitations.

3.1 Definitions

There are many criteria to evaluate performance of large distributed parallel systems, e.g. scalability, response times, first response time and throughput. Most MapReduce programs perform analysis tasks which run in background. Hence, we'll focus on execution time and throughput of the system. The model presented in this paper analyzes the execution time of each step in the framework, and the performance criterion we use is job execution time.

Definition 1. *MapReduce job execution time t is the time one MapReduce job uses to complete the computation, which starts from job submission, ends with outputting the result to the destination.*

The performance model consists of two parts: job model and system model. The job model is designed to present the computation. The system model describes the resources in the cluster that can be used in computation.

Definition 2. *A MapReduce job J is defined as a 7-tuple: (S, S', S'', X, Y, f(x), g(x)) where S is the size of input data; S' is the size of intermediate data; S'' is the size of output data; X is the number of mappers that J is divided into; Y is the number of reducers assigned for J to output; f(x) is the running time of a mapper vs. size x of input; g(x) is the running time of a reducer vs. size x of input.*

Definition 3. *A MapReduce system F is defined as a 6-tuple: (M, R, C_1, C_2, V_i, V_o, V_n), where MapReduce system F can run at most M mappers and R reducers. Each mapper has a constant overhead C_1; each reducer has a constant overhead C_2; data read rate is V_i; data output rate is V_o; and the network transfer rate is V_n. The overhead is the time spent on process initialization, and V_n is actually the data read rate from a remote node.*

Our model gives the optimal X and Y to minimize t. To simplify analysis of the model, we make the following assumptions.

1) The source data is divided and distributed equally among nodes in the cluster, and all task trackers have the same computing capacity, which is also an assumption for Hadoop design.

2) The bandwidth is enough to perform

communications at the same time without reducing the communication speed.

3) No system failure in the whole execution process.

These assumptions can reflect most systems used in small companies, where the cluster is relatively small, symmetric and stable.

3.2 Solution

There are 3 main parts of the job execution time. The parameters in model definition are shown in Figure 2.

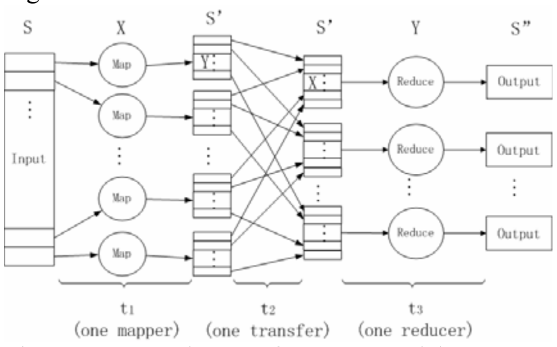

Figure 2 MapReduce performance model

t_1 is the running time of one mapper, t_2 is the time for one data transfer, and t_3 is the running time of one reducer. As system can run at most M mappers and R reducers in parallel. If X>M or Y>R, the mappers or reducers cannot completed in one round. To simplify calculation, we reduce the 3 pipelines into 2 pipelines, as shown in Figure 3.

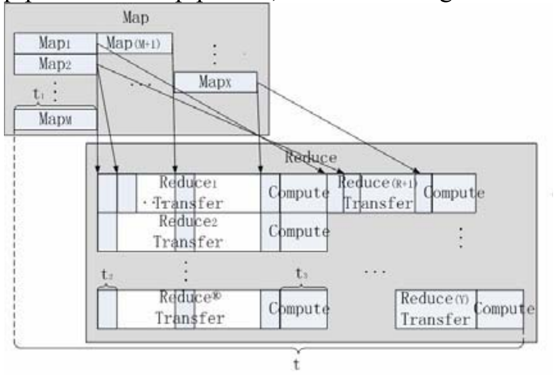

Figure 3 MapReduce execution pipeline

The first stage is the map stage. The main task is to load the data from the cluster, compute and output the intermediate results.

$$t_1 = \frac{S}{XV_i} + f(\frac{S}{X}) + \frac{S'}{XV_o} + C_1 \qquad (1)$$

As the transfer time is merged with reduce time, according to the pipelines shown in Figure 3, the transfer time shares the overhead of the reduce task. Figure 2 shows that there are XY transfers. We assume each mapper sends the same amount of data to each reducer with a data size of S'/XY, although different keys may have variable amount of values. Each reducer get X parts of the data. As the overhead C_2 is spent on initiating reducer which is before reading intermediate data from remote node, we add it on t_2 instead of t_3.

$$t_2 = \frac{S'}{XYV_n} + \frac{C_2}{X} \qquad (2)$$

In the reduce stage, the intermediate pairs are sorted by keys, processed by the reduce function and output. The running time of sorting is included in the reducer.

$$t_3 = g(\frac{S'}{Y}) + \frac{S"}{YV_o} \qquad (3)$$

According to assumption 1, all the map tasks run for the same length of time, in which case, the map tasks complete in rounds. There are λ_m rounds of map tasks; and there are M mappers complete in each round.

$$\lambda_m = \lceil \frac{X}{M} \rceil \qquad (4)$$

Similarly, there are λ_r rounds of reducers.

$$\lambda_r = \lceil \frac{Y}{R} \rceil \qquad (5)$$

There are 2 conditions according to the relationship between t_1 and t_2. Figure 3 shows the pipeline for $t_1 \geqslant Mt_2$, when reducers have to wait for data transfer to complete. When $t_1 < Mt_2$, the reducers do not need to wait for the completion of mappers except during the first round, which means there is no time intervals between intermediate data transfers in Figure 3.

$$t = \begin{cases} t_1 + \lambda_r(Xt_2 + t_3), & t_1 < Mt_2 \\ \lambda_m t_1 + Mt_2 + (\lambda_r - 1)Xt_2 + \lambda_r t_3, & t_1 \geq Mt_2 \end{cases} \qquad (6)$$

3.3 Analysis

Given a MapReduce system, and a program, we will find the optimal X and Y to minimize t. As most programs running in MapReduce are data intensive, so we can assume f(x) and g(x) are both linear. f(x)=αx and g(x)=βx respectively.

Theorem 1. $\frac{\partial t}{\partial X} < 0$ *when* $X \leqslant M$, $\frac{\partial t}{\partial Y} < 0$ *when* $Y \leqslant R$ *and* $Y \leqslant eS'$.

Proof. When $X \leqslant M$, $\lambda_m = 1$.

$$\frac{\partial t}{\partial X} = -\frac{1}{X^2}(\frac{S}{V_i} + \alpha S + \frac{S'}{V_o}) or - \frac{1}{X^2}(\frac{S}{V_i} + \alpha S + \frac{S'}{V_o} + \frac{MS'}{YV_n} + C_2)$$

$$\frac{\partial t}{\partial X} < 0$$

When $Y \leqslant R$, $\lambda_r = 1$.

$$\frac{\partial t}{\partial Y} = -\frac{1}{Y^2}(\frac{S'}{V_n} + \beta S' + \frac{S"}{V_o}) or - \frac{1}{Y^2}(\frac{MS'}{XV_n} + \beta S' + \frac{S"}{V_o})$$

$$\frac{\partial t}{\partial Y} < 0$$

So t is at minimum when $X \geqslant M$ and $Y \geqslant R$.

To simplify the analysis, real numbers are allowed in λ_m and. λ_r. So equations (4) and (5) can be simplified to X/M and Y/R. When $t_1 < Mt_2$, we get

$$X < \frac{MS'}{YV_nC_1} + \frac{MC_2}{C_1} - \frac{S}{C_1V_i} - \frac{\alpha S}{C_1} - \frac{S}{C_1V_o}$$

$$\frac{\partial t}{\partial X} = -\frac{1}{X^2}(\frac{S}{V_i} + \alpha S + \frac{S'}{V_o}) < 0$$

$$\frac{\partial t}{\partial Y} = \frac{C_2}{R} > 0$$

So $t_1 < Mt_2$ when X is small. Fig. 4 shows the function of t_1 and Mt_2 on X.

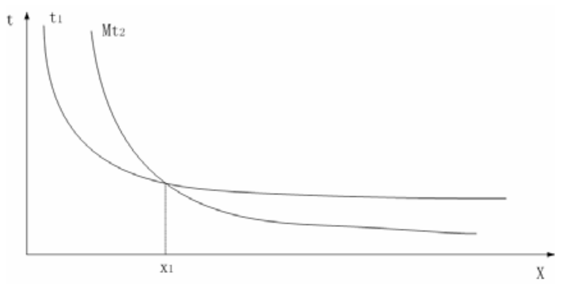

Figure 4 Relationship between t_1 and Mt_2

When $t_1 \geqslant Mt_2$,

$$\frac{\partial t}{\partial Y} = \frac{S'}{Y^2 V_n}(1 - \frac{M}{X}) + \frac{C_2}{R}$$

$$1 - \frac{M}{X} > 0; \frac{\partial t}{\partial Y} > 0$$

So t is at a minimum value when Y=R.

$$\frac{\partial t}{\partial X} = \frac{C_1}{M} - \frac{MS'}{X^2 Y V_n} - \frac{C_2 M}{X^2}$$

$$\frac{\partial t}{\partial X} = 0 => X = M\sqrt{\frac{C_2}{C_1} + \frac{S'}{C_1 R V_n}}$$

Hence, the minimum value of t occurs when

$$X = M\sqrt{\frac{C_2}{C_1} + \frac{S'}{C_1 R V_n}} \ and \ Y = R.$$

When Y remains unchanged, the relationship between X and t is shown in Figure 5. When the number of mappers is small, the program cannot perform well because of lack of parallelization. The extreme point is related to data size S'. It increases with S', i.e. more mappers are needed as data size increases.

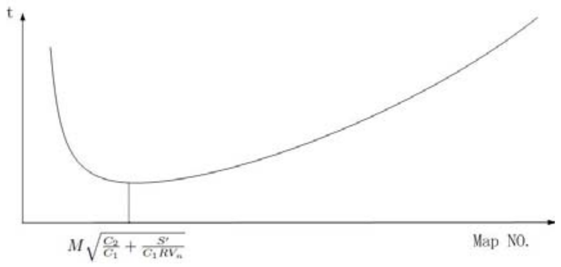

Figure 5 Relationship between number of mappers and job execution time

When X is decided, t increases as Y increases when Y>R. The relationship between t and Y is shown in Figure 6.

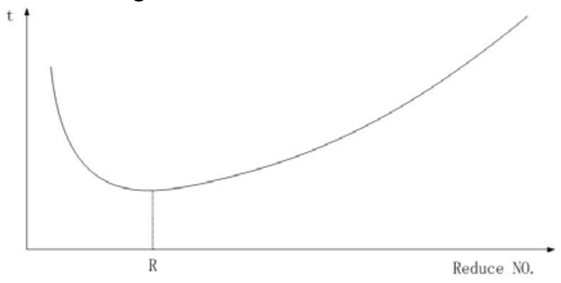

Figure 6 Relationship between number of reducers and job execution time

4 Experiments

To assess the precision of our model, we use a MapReduce implementation as the prototype. This section discusses the experimental results.

4.1 Experiment setup

We choose Hadoop as our experimental environment for its simplicity and suitability with our hardware. The cluster consists of 13 commodity computers. One of them acts as the master node, and the other 12 as slave nodes.

Hardware Environment:
Master node: RAM 8G; CPU Intel Core 2 Duo 3.00GHz, Hard Disk 160G 7200RPM; Slave node: RAM 4G; CPU Intel Pentium 4 3.00GHz; Hard Disk 1.5T 7200RPM; Switch bandwidth: 1G

Software Environment:
Operating System: Ubuntu Linux 9.10; MapReduce Implementation: Hadoop 0.20.2; System Capacity: 24 mappers and 12 reducers, 2 mappers and 1 reducer for each node.

4.2 Data set and computation task

We choose word count as our computation task, which reads a collection of files and counts how often all words occur. Each map reads documents, and produces <key, value> pairs, where the key is a specific word, and the value is 1. The reduce task adds the values with the each key to obtain a count. We prepared three tests to verify the relationships between performance and map split granularity, performance and number of reducers and data size and best map split granularity:
1) Vary map split granularity for a data size of 1G and 12 reducers.
2) Vary map split granularity for a data size of 100M and 12 reducers.
3) Vary the number of reducers for a data size of 1G and 24 mappers.
The number of map tasks cannot be configured freely in Hadoop. For the purpose of this paper, we changed the map split strategy by modifying the block size of the distributed file system.

Model parameters: M=24, R=12, V_i=42.3M/s, V_o=19.4M/s, V_n=10.8M/s, C_1=3.3s, C_2=3.2s, α=0.8s/M, β=0.9s/M. All the parameters are tested for 10 times and get the average. For wordcount program, S'≈S, and S" depends on the number of unique words, which can be assumed as a constant 1M.

4.3 Results and discussion

Figure 7, 8, 9 show the experiment results.

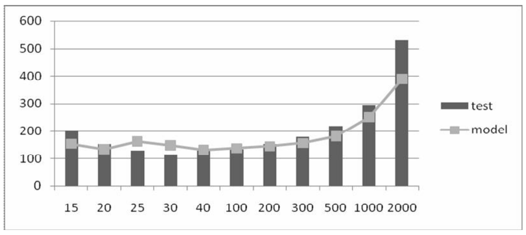

Figure 7 Relationship between X and t (1G)

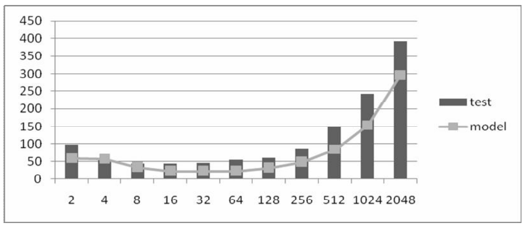

Figure 8 Relationship between X and t (100M)

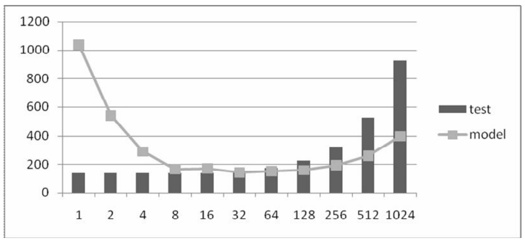

Figure 9 Relationship between Y and t (1G)

Figure 7 and Figure 8 demonstrate that the relationship between X and t match our model and the extreme point increases with data size. Figure 9 shows that there is bias in predicting the performance of reducers, because they involve sorting which is not linear as we assume in the model analysis, but the general trend is correct. In Hadoop design, the number of mappers is decided by the number of data blocks by default, which is linear with the data size. It is not optimal according to our model.

5 Conclusions

In this paper, we address the performance modeling problem for MapReduce frameworks, with the objective of better understanding the relationship between MapReduce performance and its parameters, and optimizing the overall performance (i.e. minimizing job execution time) with limited resources. We introduce an analytical performance model to analyze the execution time of each component for a MapReduce pipeline. The experimental results show that our model can predict performance and its relationship with map split granularity, number of reducers and data size. The experiment also shows that performance can be significantly improved by tuning the parameters discussed above. According to the analysis, the linear relationship between data size and number of mappers which is used in Hadoop design may not be optimal.

Acknowledgements

Supported by CHB Project “Unstructured Data Management System” (2010ZX01042-002-003).

References

[1] D. Jeffrey, S. Ghemawat. MapReduce: Simplified data processing on large clusters. Symposium on Operating System Design and Implementation, 2004
[2] Hadoop. http://hadoop.apache.org/
[3] C. Ranger, R. Raghuraman, A. Penmetsa, G. Bradski, C. Kozyrakis. Evaluating mapreduce for multi-core and multiprocessor systems. International Symposium on High Performance Computer Architecture, 2007
[4] J. Tan, X. Pan, S. Kavulya, R. Gandhi, P. Narasimhan. Mochi: Visual Log-Analysis Based Tools for Debugging Hadoop, USENIX Workshop on HotCloud, 2009
[5] S. Kavulya, et. al. An Analysis of Traces from a Production MapReduce Cluster. Carnegie Mellon University Parallel Data Lab Report CMU-PDL-09-107, 2009
[6] J. Boulon, A. Konwinski, R. Qi, A. Rabkin, E. Yang, M. Yang. Chukwa, a large-scale monitoring system. CCA, 2008
[7] G.Wang, A. R. Butt, P. Pandey, K. Gupta. A simulation approach to evaluating design decisions in MapReduce setup. MASCOTS, 2009
[8] M. Zaharia, A. Konwinski, A. D. Joseph, R. Katz, I. Stoica: Improving MapReduce performance in heterogeneous environments. OSDI'08, 2008
[9] T. Condie, N. Conway, P. Alvaro, J. M. Hellerstein, K. Elmeleegy, R. Sears. MapReduce Online. NSDI, 2010
[10] D. Jiang, B. C. Ooi, L. Shi, S. Wu. The performance of mapreduce: An in-depth study. PVLDB, 3(1):472-483, 2010
[11] T. Nykiel, M. Potamias, C. Mishra, G. Kollios, N. Koudas. Mrshare: Sharing across multiple queries in mapreduce. PVLDB, 3(1):494-505, 2010
[12] J. Berlinska, M. Drozdowski: Scheduling divisible MapReduce computations. Journal of Parallel and Distributed Computing, 71(3):450-459, 2011
[13] L. Cherkasova. Performance modeling in mapreduce environments: challenges and opportunities. The second joint WOSP/SIPEW international conference on Performance engineering, 2011