A Hadoop Performance Model for Multi-Rack Clusters

Jungkyu Han, Masakuni Ishii, Hiroyuki Makino
Distributed Computing Technology Project, Software Innovation Center
Nippon Telegraph and Telephone Corporation
Tokyo, Japan
{han.jungkyu, ishii.masakuni, makino.hiroyuki}@lab.ntt.co.jp

***Abstract*—Hadoop becomes de facto standard framework for big data analysis due to its scalability. Despite of the importance of Hadoop's scalability, there are a few works have been made on the scalability in multi-rack clusters. In multi-rack clusters of real world, network topology becomes a major scalability bottleneck due to the limited network switch capacity. It is a waste of resources to add servers to a Hadoop cluster in such situation. Therefore, it is helpful for users to save cost by efficiently measuring the network influence to Hadoop before they add a new server to their clusters. In this paper, we describe a Hadoop performance model for the multi-rack clusters. We modeled network influence on Hadoop and achieved about 95% accuracy to the real measurement. Furthermore, we predicted Hadoop scalability in large clusters with our model and show Hadoop scales enough even in multi-rack clusters.**

Keywords—Hadoop; Map-Reduce; Performance Modeling; Distributed Data Processing;

I. INTRODUCTION

As distributed analytic tools for massive data become popular, big data reveals its outstanding values. One of its contributions is for the in-depth prediction. Before the tools had not appeared to public, most of analysts could not analyze big data within a feasible time period due to limitation of computing resources. High-performance machines were expensive and only a small portion of computer experts could build distributed systems to process big data within a reasonable time period. Therefore only big companies and special facilities could analyze big data and relished gifts from the analysis.

MapReduce introduced by Google in 2004 and its de-facto open source software implementation, Hadoop [1, 2], changed the situation. Currently, many companies and researchers use Hadoop for big data analysis and present their achievements at conferences such as Hadoop World [3]. Hadoop's main characteristic is scalability. It is easily scaled out by just adding a commodity server to an existing cluster. Yahoo.Inc operates Hadoop with 3500 servers at the largest and total of 25 PB of enterprise data in 25,000 servers [4]. Despite of its scalability, an important problem is finding an efficient way to achieve better performance in a given environment. Most of newly joined users to big data analysis field are engineers in small companies or university students who do not have distributed computing experiences and sufficient budget to invest in servers. For these users, it is important to use limited resources effectively. If they predict exact Hadoop job execution time on their clusters and know the reasons, they can run the clusters more efficiently. Therefore, accurate performance model is important.

The network topology becomes a major Hadoop scalability bottleneck in real-world clusters. Generally, multi-rack configurations are used for real-world clusters because a single switch does not have a sufficient number of ports for whole severs in the cluster connect to. In the configurations, a cluster consists of several racks. Servers in a rack are connected to the rack's local switch and the local switches are connected to each other through a global switch. In most cases, port bandwidth of the global switch is similar to that of the local switch. Therefore, network bandwidth between servers in different racks is significantly small to the bandwidth between servers in the same rack. Hadoop processes are scattered over whole servers in the cluster and in 2 out of 4 processing phases; copy and reduce, the processes have to exchange data each other. Because of this reason, in industrial field, there is a question if Hadoop scales enough in real-world cluster environments. However, finding an answer to such a question is difficult because existing Hadoop performance model so far did not take the network topology into account.

We will describe a Hadoop performance model in this paper. Our performance model considers the network topology of a cluster. We examined the model's accuracy by using a cluster consists of 160 servers and 5 switches and showed it has average 5% of error. The model only needs maximum performance measured from a Hadoop job execution in a small cluster to predict execution time in larger clusters with bigger input. With our model, Hadoop users can predict job execution time on various cluster configurations or can figure out if their clusters can improve their performance before adding new servers to the cluster. In addition, by using the model, we figured out Hadoop scales well even under the real cluster conditions different from our suspicion.

In this paper, we explore current studies of Hadoop performance modeling in chapter 2. In chapter 3, we briefly

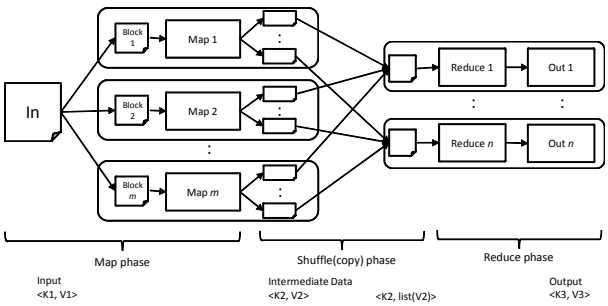

Figure 1. Map-reduce algorithm

describe Hadoop architecture to understand chapter 4 easily. We describe our model in chapter 4 and evaluate its accuracy on a cluster up to 160 nodes in chapter 5. Then we conclude our discussion in chapter 6.

II. RELATED WORK

There are several researches available for Hadoop job execution time prediction. Mumak[5] and MRperf[6] are Hadoop simulators. Mumak takes a reasonable large workload trace of real world application as input and simulates the execution time on hour scale. However, impacts of cluster size and network topology were not the primary concern of Mumak. Furthermore, in most cases, getting huge amount of real job trace is impossible for ordinary users who run small jobs. MRperf can capture network influence to the Hadoop because they incorporated their simulator with ns-2 network simulator [7]. However, MRperf did not describe performance model which helps to understand performance-architecture relationship useful for performance tuning or architecture improvement.

Hailong et al. [8] studied a statistical approach for Hadoop performance estimation. They had extracted performance critical Hadoop configurable items by statistically analyzing popular MapReduce workloads' executions and classified the workloads into groups. Then they built Hadoop performance model for each group by using machine learning techniques. Unlike our approach, Hailong's first classifies the group which includes a target Hadoop job to estimate execution time and if the job belongs to an unseen group, Hailong's approach needs machine learning to build a model for the group. Lijie et al. [9] defined and extracted Hadoop performance related characteristic values and tried to find optimal setting of the values with machine learning approaches instead of building a performance model.

Zaharia et al. [10] proposed an online formula to predict each Hadoop job's sub-task's completion time but not for the whole job. Morton et al. [11] studied a heuristic online method to predict the progress of parallel queries which can be converted into a series of Hadoop jobs. Morton's method does not consider the overlap between map and reduce stages which is important to estimate the total job execution time. Verma et al. [12, 13] studied a single job's progress in detail and presented a theoretical bound based time model to predict job's minimum and maximum duration. This method works well in homogeneous environment without high resource contention. Boduo et al. [14] built a Hadoop performance model for Hadoop cluster performance tuning to study an incremental one path analytics. However, both of Verna's and Boduo's models did not give consideration on an impact of network topology to Hadoop job execution time.

Herodotos et al. [15, 16] studied a detailed Hadoop performance model which describes dataflow and cost information at fine granularity of each phases within map and reduce tasks to build cost based Map-reduce optimizer. Although Herodotos' work considered network in their model, the work too simplified the network topology.

III. HADOOP MAP-REDUCE ARCHITECTURE

Before we discuss our performance model, we briefly describe MapReduce algorithm and Hadoop architecture.

A. Map Reduce Algorithm

As shown in Figure 1, MapReduce algorithm roughly consists of map, shuffle and reduce phases. In map phase, input data is splited to fragments which do not have mutual intersection. Then the fragments are fed into assigned map processes simultaneously. The map process executes a user defined program and generates output without interacting to other maps to relish full advantage of parallel processing. The maps' output is collected by reduces for final processing (generally, it is a summary work such as counting). This collecting phase is called reduce phase. Since reduce needs appropriate input from maps, map's output has to be properly delivered to correct reducers. This data transfer phase called shuffle phase. In MapReduce algorithm, the routing is implemented by using datakey. The algorithm assumes input is a set of records whose schema is <key1, value1>. A map process reads a record and generates output by processing the record. Map's output consists of records of <key2, value2> schema. If several kinds' output records are needed to process together in Reduce phase; the same reduce has to process them, a map assigns the same key to the output records. In shuffle phase, map output records which have the same key are sent to the same reduce. By the shuffle mechanism, each reduce has whole elements of the record sets whom the reduce needs. Therefore in the reduce phase, map outputs can be processed to generate final output pair <key3, value3>.

B. Hadoop Architecture

1) Hadoop MapReduce and HDFS

Hadoop roughly consists of two layers; Hadoop, implementation of MapReduce algorithm and its distributed file system HDFS (Hadoop Distributed File System). JobTracker and TaskTrackers are MapReduce agents. JobTracker is a conductor responsible for MapReduce job scheduling and map and reduce allocation to TaskTrackers. TaskTrackers are workers which execute map or reduce processes. HDFS also consists of two important agents. NameNode manages metadata (eg. Files list and each files' sub-block location information) and DataNodes are responsible for actual data I/O. HDFS makes one metadata file and several data blocks for a file. Data blocks distributed over DataNodes in a Hadoop cluster to allocate replicas for scalability and

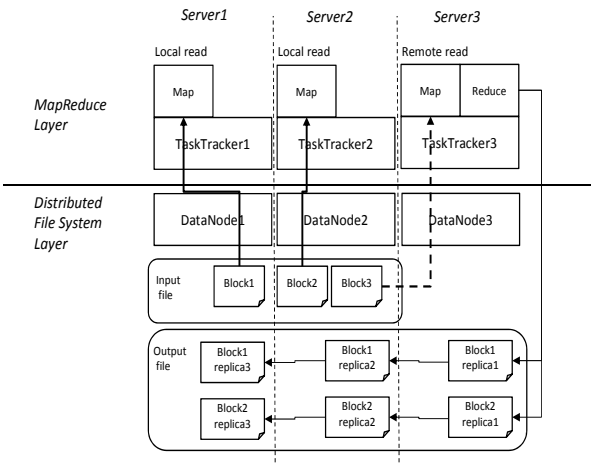

Figure 2. Map input read and Reduce output write with HDFS

availability. Hadoop closely related to HDFS with map input and reduce output. Normally, a TaskTracker and a DataNode is a pair installed in a physical server [17].

2) Map phase

In map phase, each map reads input from HDFS and generates output consist of records. Each record is sent to an appropriate reduce according to their key. The map creates several output files according to records' destination. In each map output file, records are sorted by key. If there is sufficient memory available for the data, the sort process is done in memory not to occur disk I/O. Otherwise, the file is written to the local disk (not HDFS) as a temporal fragment every time output data exceeds the available memory size. The temporal fragments are merged to create final output files before shuffle phase. Some Hadoop job uses a data combiner or compression at the sort phase to reduce network data transfer at shuffle phase. The data combiner does reduce for the map output before shuffle. For associative job like counter or average calculation, a combiner is useful to reduce the map output data size.

When a Hadoop job starts, JobTracker initiates maps on TaskTrackers for map phase. A TaskTracker can run several maps simultaneously until a number of maps matches to a predefined configuration value. It is recommended that the number is set to (a number of available cores – 1) due to the performance reason [15]. In Hadoop, one data block of input file mapped to one map. Therefore, if a number of input file blocks exceeds the number of simultaneously executable maps in a cluster, map phase will be separated into several map waves. In each map wave, the total number of executing maps is equal to the number of simultaneously executable maps in the cluster except the final wave.

To reduce network data transfer by exploiting the data locality, JobTracker tries to assign a data block to a map in the same server. The network data transfer occurs only after the attempt fails.

3) Copy phase

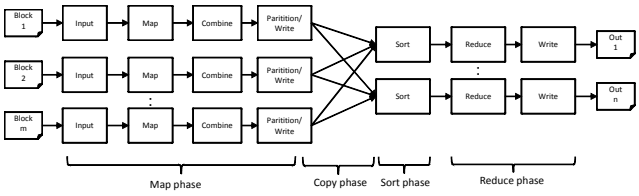

Figure 3. Hadoop Map-reduce workflow

Hadoop calls shuffle phase as copy phase. We will follow Hadoop's naming from this section. Copy phase starts from just after the first map wave finishes and ends when the whole map output data is copied to reducers. A reducer initiates this process by using plural http communication threads to connect map-running TaskTrackers. TaskTrackers have threads to interact with reducers.

4) Reduce phase

MapReduce algorithm's reduce phase is decoupled into two sub-phases in Hadoop; sort phase and reduce phase. The MapReduce algorithm requires that the whole maps' output records are grouped by their keys before reduces use them. Hadoop uses sort-by-key method to meet this requirement. The sort algorithm is the same algorithm for map output data sort at map. After the output records are sorted, the records are fed into a reduce as key and value-list pairs. Each reduce processes the records by using a user defined program. Final results are written to HDFS.

IV. PERFROMANCE MODEL

A. Assumptions

We made several assumptions for simplicity.

Assumption 1. Homogeneous and balanced cluster
Assumption 2. 2-level switch configuration
Assumption 3. Uniform data/task distribution
Assumption 4. One wave reduce phase
Assumption 5. Disabled speculative execution
Assumption 6. Map or reduce slot does not exceed number of cores in a server

In Assumption 1, we assumed that Hadoop runs in a homogeneous cluster. All servers in a cluster achieve the same performance. For example, they have the same CPU, Memory, Disk, Network interface and so on. In addition, if a cluster consists of several racks, each rack houses the same number of servers. In Hadoop community, homogeneous and balanced cluster configurations are recommended due to the performance reason [12, 17].

Then, Assumption 2 is that each rack has its own local network switch and each switch is connected to a single global switch for inter-rack communication. This assumption reflects our datacenter's typical configuration. We call the rack level switch as local switch and the single global switch as global switch in the rest of this paper. We further assumed whole local switches have the same bandwidth.

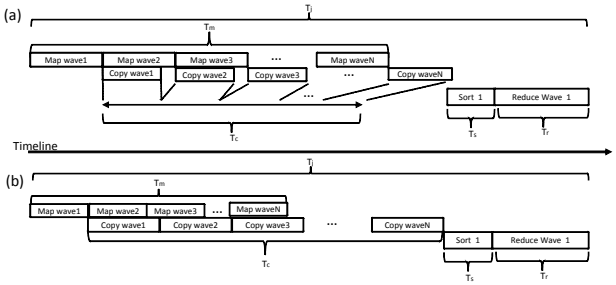

Figure 4.Map and Copy phases overlap and execution time

Assumption 3 is the generalization of Hadoop environment. HDFS tries to uniformly distribute data blocks to servers in a cluster. Therefore, in most of case, a map can find its input at the local disk with high probability. For a reduce input, a map output is almost uniformly distributed over reducers. Hadoop uses map output key hash to find targeted reduce. Therefore if the key has sufficient variable, the map output are distributed over all reduces in the cluster. Some applications such as sort are difficult to apply this assumption because each reduce needs continuous range of key for sort. However, Hadoop supports user defined routing rule. By using this rule, users may distribute maps' output to reduces almost uniformly. Hadoop also allocates reduces uniformly over servers in a cluster.

Hadoop has a configurable maximum reduce slot number which limits a number of simultaneously executable reduces in a TaskTracker. Unlike to map, user can choose a number of reduces for a job. In most cases, one wave of reduce recommended and this is our Assumption 4 [17].

Hadoop can execute duplicated map or reduce for struggler's insurance. Hadoop takes results of faster finished map and kills others. The technique called speculative execution. Our Assumption 5 prohibits this. We do not care the speculative execution because it is a fail-safe. In normal cases, fail-safe has small effect to the total execution time.

Simultaneously executing maps or reduces more than cores in a server does not have performance improvement because the server does not execute maps or reduces concurrently more than its cores. Therefore, we restrict number of concurrent maps or reduces by Assumption 6.

B. Overview

We describe our Hadoop performance model by using symbols and equations. We listed all symbols used in the equations in Table 1. In the table, the symbols use seconds for the time unit, MBytes for the size unit and Mbytes/second for the throughput (speed). Third column of Table 1 indicates symbol value's origin. 'H' means that a symbol comes from Hadoop or HDFS configuration. Each of 'C', 'A' and 'M' means its value comes from cluster environment configuration, application specific characteristics, and model respectively.

Since Hadoop consists of four stages, therefore in our model, total job execution time is represented by Equation 1. The total map execution time can be lower than just sum of whole phases' actual execution time because copy phase can be overlapped if map phase has greater than 1 wave.

$$T_j = \begin{cases} T_m + \dfrac{T_c}{N_{m\_wvs}} + T_s + T_r, & \left(T_m \geq T_c, N_{m\_wvs} \geq 1\right) \\ \dfrac{T_m}{N_{m\_wvs}} + T_c + T_s + T_r, & \left(T_m < T_c, N_{m\_wvs} \geq 1\right) \end{cases} \quad (1)$$

If a map wave's average execution time is longer than its copy time, then sum of the total map waves' execution time and a copy wave time related to the final map wave becomes map and copy phases' execution time (Figure 4 (a)). In the other hand, in reversed condition, sum of the first map wave execution time and sum of the whole copy waves' execution time becomes the map and copy phases' execution time (Figure 4 (b)).

C. Map processing time

As shown in Equation 2, we can approximate actual map phase's average execution time as multiplication of an average input block processing time and number of the total map waves in the job.

$$T_m = \frac{D_{blk\_size}}{p_m} \cdot N_{m\_wvs} \quad (2)$$

Hadoop maps one block of input files to one mapper. If the total number of blocks is greater than the cluster's total map slots, then map phase will be divided several waves according to Equation 3. Ceiling is used in equations in this paper to get the slowest member's finish time in a cluster for calculating group finish time of distributed system.

$$N_{m\_wvs} = \left\lceil \frac{max\left(N_{m\_slots}, \left\lceil \frac{N_{in\_blks}}{N_{servers}} \right\rceil\right)}{N_{m\_slots}} \right\rceil \quad (3)$$

In this paper, we got an average map throughput p_m from actual measurement of a real job's map input block size and average processing time. Since we use throughput from real measurement, we do not care about combine and compression cost in Equation 3. Unlike reduces, maps' input (data block size) is always fixed regardless environments such as the total input size because of HDFS configuration. Therefore, combine or compression cost is nearly same over whole maps under our assumptions. In addition the costs are included in p_m.

Each map wave finishes after the last map in the wave finishes. We can exploit this to calculate maximum map phase execution time bound by using mostly least map throughput that we observed during measurement. However, in this paper, we deal with an average case because the case is a start point to evaluate performance.

D. Copy processing time

TABLE I. EQUATION SYMBOLS

Symbol	*Expression*	*Source*
T_j	Job execution time.	M
T_m	Accumulated actual map wave execution time	M
T_c	Accumulated actual copy wave execution time	M
T_s	Actual sort execution time	M
T_r	Actual reduce execution time	M
T_{r_rep}	Total reduce output writing time	M
T_{r_proc}	Total reduce processing time	M
N_{m_wvs}	Number of map waves	M
N_{m_slots}	Number of simultaneously executable maps in a server	H
N_{r_slots}	Number of simultaneously executable teducers in a server	H
$N_{servers}$	Number of servers in a Cluster	C
D_{blk_size}	Data block size	H
N_{in_blks}	Number of map input data blocks	A
p_m	Average map throughput per server core	A
p_r	Average reduce throughput per server core	A
N_{racks}	Number of racks in a cluster	C
$N_{reduces}$	Number of reduces	A
$N_{cp\_threads}$	Number of copy i/o threads per reducer node	H
B_{local}	Network Bandwith of intra-switch connection	C
B_{remote}	Network Bandwith of inter-switch connection	C
S_{cp_thread}	Theoretical maximum copy speed per copy thread	H
S_{r_rep}	Theoretical maximum output replication speed per reduce	H
S_{disk_write}	Sequential disk write speed	C
r_{map_io}	Map output to input size size ratio	A
r_{reduce_io}	Reduce output to input size ratio	A
$N_{replicas}$	Number of replicas in HDFS	H
N_{rep_locals}	Number of local rack replicas $N_{replica} = 1 + N_{rep\_locals} + N_{rep\_remotes}$	H
$N_{rep\_remotes}$	Number of local rack replicas $N_{replica} = 1 + N_{rep\_locals} + N_{rep\_remotes}$	H
$N_{sortpaths}$	Number of merge sort paths for Copy.	H
$D_{sortbuf}$	Sort buffer size for copy	H

H:Hadoop/HDFS configurations, C:Cluster configurations,
M:Model-generated value, A:Application specific characteristics

Since copy phase uses network data transmission, we should consider the network topology to achieve better performance estimation. The primal factor of the performance bottleneck is inter-switch bandwidth (local switch to global switch and vice versa) because in most cases, the bandwidth is lower than intra-switch (from server to server. and both are physically connected to a same local switch) bandwidth.

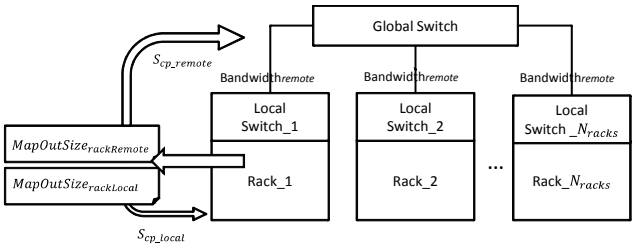

Figure 5. 2 types of map output data classified by its target

Suppose that we run a Hadoop cluster consists of N_{racks} racks and each rack houses (N_{nodes}/N_{racks}) servers. In our assumption, map output is uniformly distributed to whole reduces and reduces are also randomly placed on among servers in the cluster. Therefore, like Figure 5, map output in a rack can be classified by targeted reduces location; Data for reduces which are placed in the same rack or others. We call data copy to reduces of same rack as local copy and copy to reduces of remote lacks as remote copy. Basically, as shown in Equation 4, copy execution time T_c is governed by maximum copy time between local copy and remote copy. In addition, copy execution time is the value of map output data divided by copy speed. We can calculate average total map output data size generated from a rack by using Equation 5.

$$T_c = max\left(\frac{MapOutSizeForLocalReduce}{LocalRackCopySpeed}, \frac{MapOutSizeForRemoteReduce}{RemoteRackCopySpeed}\right) \quad (4)$$

$$MapOutSizeForAllReduce = \frac{N_{in\_blks} \cdot D_{blk\_size} \cdot r_{map\_io}}{N_{racks}} \quad (5)$$

$$MapOutSizeForLocalReduce = \frac{MapOutSizeForAllReduce}{N_{racks}} \quad (6)$$

$$MapOutSizeForRemoteReduce = MapOutSizeForAllReduce \cdot \left(1 - \frac{1}{N_{racks}}\right) \quad (7)$$

In Equation 5, size of map output data generated from a rack is calculated dividing total map output data by number of racks. Map's input data size can be represented by multiplication of HDFS block size and a number of a job's input data blocks in HDFS. Since input data size changes after map, we multiplied map output to input ratio r_{map_io} to the input data size. The ratio is chosen from a real job execution observation. 1/ N_{racks} of total map output data from a rack is used for local copy and other portion of data is used for remote copy as shown in Equation 6, 7 respectively. Because we took uniform distribution assumptions of map output destinations (reduces) and reduce location. Furthermore, we have N_{racks} racks in the cluster.

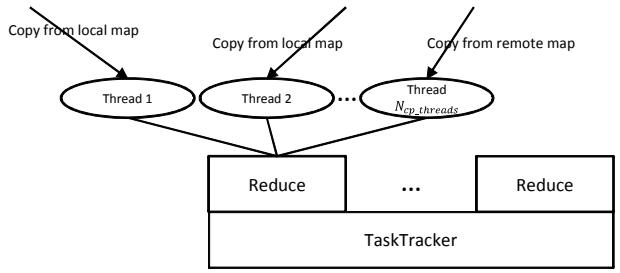

Figure 6. Map output copy threads of a reduce server

To calculate copy execution time T_c, we also had to know local and remote copy speed. In most cases, local copy speed is limited by local bandwidth and remote copy speed is limited by remote bandwidth.

However when a number of reducers is small, Hadoop architecture may limit copy speed. Showed in figure 6, a reduce has plural threads dedicated for map output data copy. In copy phase, the threads establish a connection to a map server to get map output. Therefore in a reduce's perspective, theoretical maximum copy speed is the sum of its copy threads' speed. In reality, the thread's copy speed is limited to minimum value between thread's theoretical maximum copy speed and actual copy speed limited by local bandwidth as shown in Equation 9. In Equation 9, we get the actual copy speed for a thread by dividing a reduce running server's local bandwidth by a number of the concurrent copy threads in the server. We got the number of the concurrent copy threads with Equation 8. Equation 8 approximates number of concurrent reduce per server at a given time point is minimum of reduce slot number for a server and quotient of total reduce number divided by node number.

$$ConcurrentReducesPerServer = \min\left(N_{r\_slots}, \left\lceil \frac{N_{reduces}}{N_{servers}} \right\rceil\right) \tag{8}$$

$$CopyThreadSpeed = \min\left(\frac{B_{local}}{\min\left(N_{r\_slots}, \left\lceil \frac{N_{reduces}}{N_{servers}} \right\rceil\right) \cdot N_{cp\_threads}}, S_{cp\_thread}\right) \tag{9}$$

Like Equation 10, a rack's local copy speed is calculated by multiplication of a thread's copy speed of Equation 9 and number of concurrent local copies in the rack.

$$LocalRackCopySpeed = CopyThreadSpeed \cdot ConcurrentLocalCopies \tag{10}$$

We can estimate the number of concurrent local copies in a rack by multiplying the number of reduces' threads in the rack and their probability of local copy use. Equation 11 calculates the number of concurrent copy threads in the rack by multiplication of the rack's number of reduces and the Hadoop's number of concurrent copy threads per reduce. Addition, Equation 11 also calculate the threads local copy probability by dividing the number of rack's concurrent copy threads by the number of racks in the cluster because maps are distributed all servers in the cluster and each thread has equal chance to connect to the each map.

$$\begin{aligned} &ConcurrentLocalCopies \\ &\quad = NumberOfReduceInLocalRacks \cdot NumberOfThreadPerReduce \cdot ProbailityOfLocalCopy \\ &\quad = \frac{N_{reduces}}{N_{racks}} \cdot N_{cp\_threads} \cdot \frac{1}{N_{racks}} \end{aligned} \tag{11}$$

For a given rack, a number of concurrent remote copy threads which are connected to the rack's servers from other racks', is calculated by Equation 12. The equation multiplies the number of reducers in other racks, the Hadoop's number of concurrent copy threads per reduce and the threads' remote copy probability. The number of reducers in other racks and the probability can be calculated in a similar manner to local copy probability.

$$\begin{aligned} &ConcurrentRemoteCopies \\ &\quad = NumberOfReducerInRemoteRacks \cdot NumberOfThreadPerReduce \cdot ProbailityOfRemoteCopy \\ &\quad = N_{reduces} \cdot \left(1 - \frac{1}{N_{racks}}\right) \cdot N_{cp\_threads} \cdot \left(1 - \frac{1}{N_{racks}}\right) \end{aligned} \tag{12}$$

Equation 13 reflects the fact that the rack's local copy speed, the remote bandwidth limits the rack's remote copy speed unlike the rack's local copy speed.

$$\begin{aligned} &RemoteRackCopySpeed \\ &\quad = min(RemoteNetworkBandwidth, CopyThreadSpeed \cdot ConcurrentRemoteCopies) \end{aligned} \tag{13}$$

$$RemoteNwBandwith = B_{remote} \tag{14}$$

For more depth study, we should consider copy from a map running TaskTracker's (server's) perspective. At the TaskTracker, there are both connections from local reduces and remote reduces and the connections contend for the TaskTracker's network bandwidth. However, in real cases, contention does not occur because the remote copy speed is so small compared to the local copy speed and local bandwidth is large enough.

E. Sort processing time

For sort phase execution time estimation, we followed Boduo's study[14] because sort phase is not related to network. Furthermore, Boduo gave detailed sort phase performance equation in their study. Accroding to Boduo's study, sort phase processing time can be estimated by Equation 15. In Equation 16, we substitute symbols of the Boduo's reduce part's sort execution time equation with ours.

$$\lambda_F(n, b) = \left(\frac{1}{2F(F-1)} n^2 + \frac{3}{2} n - \frac{F^2}{2(F-1)}\right) \cdot b \tag{15}$$

$$T_s = \begin{cases} \dfrac{2 \cdot \left\lceil \frac{N_{reduces}}{N_{servers}} \right\rceil}{S_{disk\_write}} \cdot \lambda_{N_{sortpaths}}\left(\dfrac{N_{in\_blks} \cdot D_{blk\_size} \cdot r_{map\_io}}{N_{reduces} \cdot D_{sortbuf}}, D_{sortbuf}\right), & \dfrac{N_{in\_blks} \cdot D_{blk\_size} \cdot r_{map\_io}}{N_{reduces} \cdot D_{sortbuf}} > 1 \\ 0, & otherwise \end{cases} \tag{16}$$

F. Reduce processing time

To model reduce phase performance, we should again consider network topology because reduce results are normally written to HDFS which enables replication over servers in a cluster for availability reason. After sort process finished, the

sorted data is used as reduce process's input. A reduce generates output data by processing sorted data. The output data is written to HDFS as soon as it is generated. Processing and writing are done simultaneously. Since we assumed reduces are uniformly distributed over all servers in a cluster, we can estimate reduce processing time by calculating an individual reduce server's average execution time. Like Equation 17, the server's reduce execution time is determined by taking maximum value between reduce processing time and output data replication time.

$$T_r = \max(T_{r\_proc}, T_{r\_rep}) \tag{17}$$

Reduce processing time on the server can be calculated by multiplication of single reduce processing time and number of reduce waves as shown in Equation 18. Since we assumed a single reduce wave environment, a number of reduce waves is 1.

$$\begin{aligned} T_{r\_proc} &= SingleReduceProcessingTime \cdot NumberOfReduceWaves \\ &= \frac{N_{in\_blks} \cdot D_{blk\_size} \cdot r_{map\_io}}{N_{reduces} \cdot p_r} \cdot \left\lfloor \frac{N_{reduces}}{N_{servers} \cdot N_{r\_slots}} \right\rfloor \end{aligned} \tag{18}$$

HDFS makes 3 replicas of each data block (including original data) and deploys them over servers in a cluster. HDFS exploits a multi-rack environment condition to achieve higher availability. For example, a replica is stored in a local server which initiated the data write operation. One of other replicas is sent to a server in the same rack and last one is sent to a server of different rack. The replicas are called local and remote replication respectively. The replication process uses bucket relay method. The process also has available bandwidth limit of replication to avoid network resource starvation.

$$T_{r\_rep} = \begin{cases} Max(T_{rep\_disk\_write}, T_{rep\_local\_rack}), & G_n = 1 \\ Max(T_{rep\_disk\_write}, T_{rep\_local\_rack}, T_{rep\_remote\_rack}), & G_n > 1 \end{cases} \tag{19}$$

Equation 19 represents replication execution time model of a single rack cluster and a multiple rack cluster. Replication time is equal to the minimum of disk writing time and replication time. In one-rack cluster environment, we just consider local replication only. However, in multi-rack environment, we should care of remote replication. In this paper, we describe replication up to 3 replica cases because 3-replica-configuration is the most popular configuration in the Hadoop community [17].

$$\begin{aligned} T_{rep\_disk\_write} &= \left(\frac{ReplicationDataSizePerHDFSDataServer}{DiskWriteSpeed} \right) \\ &= \frac{N_{in\_blks} \cdot D_{blk\_size} \cdot r_{map\_io} \cdot r_{reduce\_io} \cdot N_{replicas}}{N_{servers}} \cdot \frac{1}{S_{disk\_write}} \end{aligned} \tag{20}$$

Reduce output data is uniformly distributed over servers in the cluster. Therefore for a given server, we can calculate writing data size by multiplication of number of replicas and reduce output data size divided by number of nodes in the cluster. As shown in Equation 20, we can get reduce disk writing time by dividing the reduce output size by disk sequential writing speed. We can get the writing speed by running disk benchmark program.

Since one reduce makes one output file, the cluster has $N_{reduces}$ replication chains totally. Most of replicas except for one which is written to local server use network. Therefore, the total number of replicas using network is multiplication of $N_{reduces}$ and ($N_{replicas}$ - 1). If a cluster consists of a single rack, whole network using replications are local replication. Since replication speed is limited by a server's local network bandwidth, we should focus on that how many concurrent replications the single has.

With Equation 8, we approximated a number of concurrent reduces per server at a given time point.

$$OutputSizePerReduce = \frac{N_{in\_blks} \cdot D_{blk\_size} \cdot r_{map\_io} \cdot r_{reduce\_io}}{N_{reduces}} \tag{21}$$

$$T_{rep\_local\_rack} = \frac{OutputSizePerReduce}{LocalReplicationSpeedPerReduce} \tag{22}$$

$$\begin{aligned} &LocalReplicationSpeedPerReduce \\ &= \min\left(ReplicationSpeed, \frac{LocalBandwidth}{ConcurrentReducesPerServer \cdot LocalReplicas} \right) \\ &= \min\left(S_{r\_rep}, \frac{B_{local}}{\min\left(N_{r\_slots}, \left\lceil \frac{N_{reduces}}{N_{servers}} \right\rceil \right) \cdot N_{rep\_locals}} \right) \end{aligned} \tag{23}$$

For a single server's perspective, it performs N_{rep_locals} of concurrent reduce replications at a given time point because its role is both a replication initiator and a replication relayer for other servers. And as mentioned before, the replication channels share the server's local network bandwidth. Therefore, average replication speed is the minimum number of pre-defined replication speed limit and available bandwidth value. In Equation 23, we got the available bandwidth by dividing the server's local bandwidth by the active replication channel number. In the local replication case, the active replication channel number is multiplication of the number of concurrent reduces per server and a number of local replicas.

$$T_{rep\_remote\_rack} = \frac{\text{OutputSizePerReduce}}{\text{RemoteReplicationSpeedPerReduce}} \tag{25}$$

$$\begin{aligned} &RemoteReplicationSpeedPerReduce \\ &= \min\left(ReplicationSpeed, \frac{RemoteBandwidth}{ConcurrentReducesPerServer \cdot ReduceNodesInARack \cdot RemoteReplicas} \right) \\ &= \min\left(S_{r\_rep}, \frac{B_{remote}}{\min\left(N_{r\_slots}, \left\lceil \frac{N_{reduces}}{N_{servers}} \right\rceil \right) \cdot \min\left(\frac{N_{reduces}}{N_{racks}}, \frac{N_{servers}}{N_{racks}} \right) \cdot N_{rep\_remotes}} \right) \end{aligned} \tag{26}$$

Equation 25 and 26 represents remote replication time calculation. The differences from the local replication time are that remote replication uses remote bandwidth instead of local bandwidth and considers whole servers' replication in a rack because a path from a local switch to the global switch is shared among whole servers in the rack.

V. MODEL EVEALUATION

A. Environment and parameter setting

To evaluate our model's correctness, we set up a cluster which had up to 160 servers and ran our MapReduce program on the cluster. The cluster consisted of 4 racks and each of them had 40 servers. Each rack had its own local switch. And all servers in the rack connected to the switch. The local switches were connected to a global switch to communicate each other. We installed Hadoop TaskTracker and HDFS DataNode in every server of the cluster. We prepared

TABLE II. CLUSTER SPECIFICATION

Server	CPU	Intel Xeon X3430 2. 4Ghz (4Core)
	Memory	8GB
	Disk	250GB HDD × 2
	NIC	Gigabit Ethernet
Software	OS	Linux CentOS 5.5
	Hadoop	Apache Hadoop 0.21.0
	Java	Sun JDK 6u21
Switch	Global	Cisco Catalyst 3750G
	Local	Cisco Catalyst 2960-S
Rack	Quantity	4
	Nodes/Rack	40 nodes

dedicated servers and network channel for JobTracker and NameNode to keep them interfering from MapReduce data operation. In summary, we used 162 Servers and 5 local network switches in total. All servers had same specification and software configuration. We listed our cluster's detailed specification in Table 2.

Our model equation needs values from Hadoop, HDFS, a cluster and a MapReduce application. We summarized our experiment's Hadoop, HDFS, cluster values in Table 3. Benchmark programs were used for disk and network performance measurement. We used Bonnie++ [18] for disk I/O performance and Netperf[19] for network bandwidth.

B. Parameter measurement from real application running

We ran our MapReduce program as a model validation program. The program reads records from input files and encrypts the each record in map phase, then sorts and writes map result to HDFS as output files in reduce phase. The program is easy to observe copy and reduce phase's network affection because the program's map output data size is sufficiently big. In addition, since the program does not need to be run in a single reduce configuration, we can observe multi-reduces' contribution to Hadoop.

Our model also needs application specific parameters. We got the values from trials of various configurations in a small cluster. The cluster should be one wave map phase with full occupied map slot to get exact copy phase execution time. In addition, sufficient number of map or reduce servers are required to examine performance limit of each phase's process. In our case, we used two configurations; 1 server for maps and 4 servers for reduces and vice versa. We listed the parameters from the trial in Table 4.

In the next section, we will compare our model's predicted values to the values from our real application running measurements under various configurations. To get the real measurement value, we used an on-line Hadoop Map-reduce progress report that users can see when they executes MapReduce job. The report periodically informs map's and reduce's progress in percentage manner. For map reports, 100% means whole maps are finished. For example, if a job has 2 map waves, and 1 map wave just finished, then the progress is 50%. In reduce phase, progress is divided by 3 parts. Roughly from 0% to 33% represents copy phase progress, from 34% to 66% shows sort phase progress, and final part means reduce phase progress. Since Hadoop gets this progress from individual Tasktrackers and reports average to user, the report has some error compared to real progress. However, we used this value because users get this report easily and our results showed our model still could grasp Hadoop's scalability despite of this kind of error.

TABLE III. CLUSTER SPECIFICATION

Module	Symbol	Hadoop/HDFS Configure	Value
Hadoop	N_{m_lots}	mapreduce.job.maps	3
	N_{r_slots}	mapreduce.job.reduces	3
	$N_{cp\_threads}$	mapred.reduce.parallel.copies	5
	S_{cp_thread}	N/A	10
	$N_{sortpaths}$	io.sort.factor	10
	$D_{sortbuf}$	[Heap size of a reduce] X mapred.job.shuffle.input.buffer.percent	140
HDFS	D_{blk_size}	block.size	64
	S_{r_rep}	N/A	10
	$N_{replicas}$	dfs.replication	3
	N_{rep_locals}	N/A	1
	$N_{repl\_remotes}$	N/A	1
Cluster	N_{nodes}	N/A	20~160
	N_{racks}	N/A	1~4
	B_{local}	N/A	100
	B_{global}	N/A	100
	S_{disk_write}	N/A	66

TABLE IV. APPLICATION SPECIFIC PARAMETERS

m_{thput} (MB/s)	0.4884
r_{thput}(MB/s)	10
m_{io_ratio}	2.447
r_{io_ratio}	1.0

C. Experimentand result

We ran the application in 9 different cluster configurations to investigate Hadoop's scalability characteristics and compared them to our model's prediction. We designed cases 1 to 4 to evaluate our model's scalability under a single rack condition. From case 1 to 2, input size is fixed and only a number of servers in the cluster increased from 20 to 40. From case 2 and 3, only the number of reduces increased from 20 to 40. Therefore, we expected the job execution time decrease from case 1 to 3. Case 4 had the same settings to the case 3 but doubled input size. Therefore, if Hadoop scales well, Job execution time will just be doubled in the case 4 compared to the case 3. Case 5, 6, 7s purpose was the same to the case 2, 3, 4. Only difference was doubled number of racks. Similarly, case 8 and 9's purpose was same to the case 2 to 3 except the number of racks.

We used case 4 and 5, 7 and 8 to examine Hadoop's scalability at the rack increasing point. Therefore, only the number of racks, thus the number of servers changed from 40

TABLE V. COMPARISON BWETWEEN REAL MEASUREMENT AND MODEL

Cases		1	2	3	4	5	6	7	8	9
N_{racks}		1	1	1	1	2	2	2	4	4
N_{nodes}		20	40	40	40	80	80	80	160	160
$N_{reduces}$		20	20	40	40	40	80	80	80	160
N_{in_blks}		120	120	120	240	240	240	480	480	480
T_m	R	262	130	131	262	131	129	335	130	130
	M	262	131.0	131.0	262.1	131.0	131.0	262.1	131.0	131.0
T_c	R	138	22	11	138	80	84	217	138	130
	M	18.0	18.0	9	18.0	93.9	93.9	187.9	140.9	140.9
T_s	R	22	26	22	37	36	17	39	56	22
	M	22.1	22.1	2.3	22.1	22.1	2.26	22.13	22.1	2.3
T_r	R	99	78	52	103	242	178	370	215	180
	M	94.0	94.0	47	94.9	187.9	187.9	375.85	187.9	187.9
T_j	R	393	257	216	411	489	408	760	538	462
	M	387.2	265.1	189.3	387.2	435.1	415.2	754.0	482.0	462.2

to 80 at the case 4 to 5 and 80 to 160 at the case 7 to 8. If Hadoop has ideally linear scalability, the job execution time becomes half at each transition.

Table 5 shows all of settings and results. In Table 5, 'R' means real measurement of execution time and 'M' means our model predicted execution time. A comparison graph in Figure 7 plots Table 5's job execution time T_j of 'R' and 'M' in Table 5. 'R' and 'M' shows the same pattern. And an average percentage of value difference 'R' from 'M' to 'R' is 5.2% and standard deviation of the percentage is 4.86. This shows that our model can describes real job's execution time under multi rack conditions.

From case 1 to 2, our model showed map execution time became half but reduce execution time is same because input size and number of reducers does not change. From case 2 to 3, execution time decreased. As we expected, from 3 to 4, the job execution time roughly doubled because input data size was doubled while a number of reducers was fixed. This means Hadoop scales well in the single rack environment.

An interesting point is that case 5 was slower than case 4 even the number of servers were doubled therefore map phase execution time became half. The result is due to remote copy cost which introduced first from case 5. Our prediction shows that case 5's Copy execution time is almost 4.5 times longer compared to case 4. However, real measurement showed that case 4's copy time is longer than case 5. This was caused by two wave map phase execution of case 4. As shown in figure 4, copy starts immediately after first map wave finishes. Then Hadoop waits for second map wave's finish before actual second copy starts. The Hadoop Mapreduce progress report starts counting copy time when the first copy wave starts and finishes when the last copy wave ends. For this reason, observed copy time is longer than prediction under multi map wave conditions even if actual copy size is short. All multi wave trial (case 1, 4, 7)'s copy time supports this.

Within the 2 rack condition (case 5, 6, 7), Hadoop's scalability was similar to what we observed in 1 rack cases.

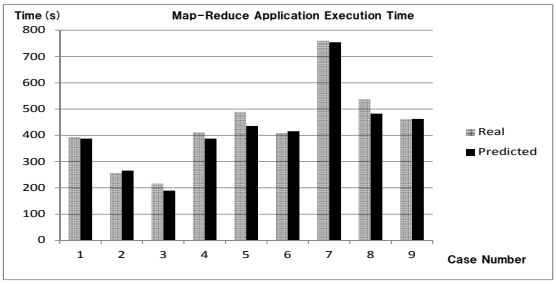

Figure 7. Comparison bwtween real measurement and model prediction

From 2 rack configuration to 4 rack configuration transition (from case 7 to 8), unlike 1 rack to 2 rack transition, It scaled out well. Interesting point is case 8's reduce phase time decreased even number of inputs and reducers were same between case 7 and 8. This can be explained increased available inter-switch bandwidth for each reducer. Because the number of racks and servers in the cluster doubled, the number of reducers which were placed per rack was decreased, therefore, inter switch bandwidth contention for replication was relieved.

From case 8 to 9 only a number of reducers doubled, however there were no significant performance improvement. Our prediction results shows only sort phase has significant improvement. This implies 40 reduces in case 8 already hit inter-node connection limit. This reveals that under multi-rack condition reduce phase has early scalability limit before severs capacity limit due to network limitation.

D. Scale out prediction

Under multi-rack condition, the most important factor for scalability is inter-switch bandwidth between local switch and global switch. We used our model to predict our sample application's scalability in various inter-switch bandwidth conditions.

We defined 4 global network conditions according to inter-switch bandwidth from 100 MB/s to 400 MB/s. And under each condition, we defined sub-cases according to a number of the cluster's racks from 1 to 32 racks. In each case, we fixed input data to 3840 blocks. The purpose of this prediction was examine if Hadoop really scales out under a cluster consists of many racks.

We plotted our prediction in Figure 8. According to our model's prediction, a 400 MB/s of inter-network bandwidth cluster required to get ideal scale out performance under our configuration. However, even in a 100 MB/s inter-network bandwidth, a typical cluster for most users, Hadoop was scalable. In the previous section, we noticed our application did not scale out well at one rack to two rack cluster size increment due to inter-network limitation. However, Figure 8 showed that from two rack configuration to any more rack configuration, Hadoop scales out well. This was caused by combination of input data and reducers' uniform distribution (if our assumption holds). Therefore, according to Equation 4, 5, 7, the rack's map output size for remote copy decreases as

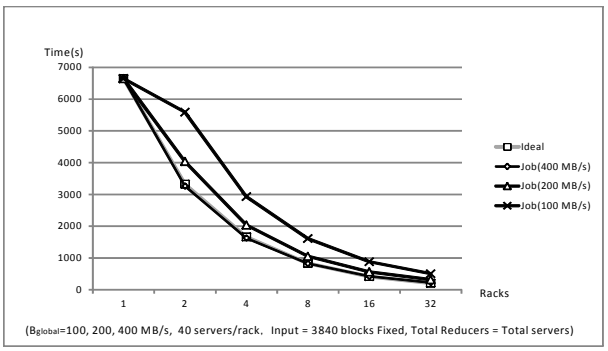

Figure 8. Job execution time prediction under various inter-switch bandwidth conditions

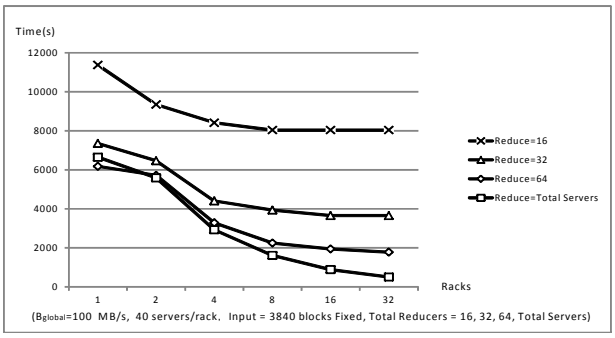

Figure 9. Job execution time prediction under various number of reducers

numbers of racks increase, even the rack's remote copy speed in Equation 12, restricted to inter-switch bandwidth of Equation 13.

In some cases, a number of reduces is more important than a number of maps. As shown in equation 14, 15 reduce's sort time is proportional to square of reducer's input data size. Therefore, if number of reduces is small under huge map input conditions, map's scalability does not contribute to Hadoop's scalability. In Figure 9, we plotted a job's execution time prediction under various numbers of reducer conditions. Under the 16 reduce condition, Hadoop's scalability limits at 8 rack configurations. The limits glows as a number of reducers glows. This clearly shows the importance of concurrent reduce number. However it is difficult for some algorithm to design distributed reduces. Therefore, implementing a distributed sorting over servers will be helpful for improve such algorithm's scalability because sort itself does not affect to the algorithm's context.

VI. CONCLUSION

In this paper, we suggested a network aware Hadoop performance model and explored its scalability with the model. Our experimental results showed that the model could predict a job's execution with 5 % average error compared to the real job measurement within up to 4 rack 160 server clusters in resolution of second. We could explain the cause of the real job measurement pattern under various conditions with the model. In addition, with the model, we explored the Hadoop scalability characteristic in large clusters under various inter-switch bandwidths and a numbers of reduces which are important for the scalability. By using the result, we showed Hadoop scales out well in the typical cluster network configurations, and suggested guidelines for application design and Hadoop scalability improvement.

A 160 server cluster is a sufficiently large cluster for most of newly joined users to big data analysis without sufficient budget for cluster and experience. Our model contributes the users to save cost by assisting to analyzing potential of their cluster before adding a new server.

REFERENCES

[1] J. Dean and S. Ghemawat, "MapReduce: Simplified data processing on large clusters," Proc. of OSDI'04, pp. 137-150, Dec. 2004.

[2] The Apache Software Foundation, "Hadoop Map-Reduce tutorial," Hadoop Project

[3] Hadoop world, http://www.hadoopworld.com/

[4] K. Shavachko, H. Kuang, S. Radia and R. Chansler, "The Hadoop distributed file system," Proc. of MSST 2010, pp.1-10, 2010.

[5] Apache Hadoop MapReduce: Mumak: Map-Reduce simulator. Available online at https://issues.apache.org/jira/browse/MAPREDUCE-728. Accessed on Nov. 2012

[6] Guaying Wang, Ali R. Butt, Prashant Pandey, Karan Gupta, "Using realistic simulation for performance analysis of mapreduce setups", Proc. of LASP 09. Pp.19-26, 2009

[7] Ns-2, http://www.isi.edu/nsnam/ns/, Accessed on Nov.2012

[8] Hailong Yang, Zhongzhi Luan, Wenjun Li, Depei Qian, "MapReduce Workload Modeling with Statistical Approach", Journal of Grid Computing, Volume 10, Issue 2, pp.279-310, June 2012

[9] Lijie Xu, "MapReduce Framework Optimization via Performance Modeling", Proc. of Parallel and Distributed Processing Symposium Workshops & PhD Forum (IPDPSW), 2012

[10] M. Zaharia, A. Konwinski, A.D. Joseph, R. Katz, and I.Stoica. "Improving MapReduce performance in heterogeneous environments", Proc. of OSDI, 2008

[11] K. Morton, A. Friesen, M. Balazinska, and D.Grossman. "Estimating the progress of MapReduce pipelines", Proc. of ICDE, 2010.

[12] A. Verma, L. Cherkasova, and R. Campbell. "ARIA:Automatic Resource Inference and Allocation for MapReduce Environments," Proc. of ICAC, 2011.

[13] A. Verma, L. Cherkasova, and R. Campbell. "Resource Provisioning Framework for MapReduce Jobs with Performance Goals," Proc. of Middleware, 2011.

[14] Boduo Li,Edward Mazur,Yanlei Diao,Andrew McGregor,Prashant Shenoy, "A platform for scalable one-pass analytics using MapReduce" Proc. of the 2011 ACM SIGMOD International Conference on Management of data. pp. 985-996, 2011

[15] Herodotou, H., Babu, S. "Profiling, What-if Analysis, and Cost-based Optimization of MapReduce Programs," Proc. of VLDB, 2010.

[16] Herodotos Herodotou, "Hadoop Performance Models", Technical Report, CS-2011095, Duke University, http://arxiv.org/abs/1106.0940, 2011

[17] Tom White. 2010. "Hadoop:The Definitive Guide" (Second edition). O'Reilly Media/Yahoo Press

[18] Bonnie++, http://sourceforge.net/projects/bonnie/, Accessed on Nov.2012

[19] Netperf, http://www.netperf.org/netperf/, Accessed on Nov.2012