

# MR-Scope: A Real-Time Tracing Tool for MapReduce

Dachuan Huang, Xuanhua Shi, Shadi Ibrahim, Lu Lu, Hongzhang Liu, Song Wu, Hai Jin  
Cluster and Grid Computing Lab  
Services Computing Technique and System Lab  
Huazhong University of Science and Technology, Wuhan, 430074, China  
hjin@hust.edu.cn

## ABSTRACT

MapReduce programming model is emerging as an efficient tool for data-intensive applications. Hadoop, an open-source implementation of MapReduce, has been widely adopted and experienced by both academia and enterprise. Recently, lots of efforts have been done on improving the performance of MapReduce system and on analyzing the MapReduce process based on the log files generated during the Hadoop execution. Visualizing log files seems to be a very useful tool to understand the behavior of the Hadoop process. In this paper, we present MR-Scope, a real-time MapReduce tracing tool. MR-Scope provides a real-time insight of the MapReduce process, including the on-going progress of every task hosted in Task Tracker. In addition, it displays the health of the Hadoop cluster data nodes, the distribution of the file system's blocks and their replicas and the content of the different block splits of the file system. We implement MR-Scope in native Hadoop 0.1. Experimental results demonstrate that MR-Scope's overhead is less than 4% when running wordcount benchmark.

## Categories and Subject Descriptors

D.4.7 [Operating System]: Organization and Design—*Distributed Systems*. D.4.8 [Operating System]: Performance—*Monitors*.

## General Terms

Design, Management.

## Keywords

MapReduce, Hadoop, Real-time Tracing, Visualization.

## 1. INTRODUCTION

With the emergence of cloud computing in association with the incredible increasing of data, how to efficiently process huge amount of data has become a challenging issue. MapReduce [1] is currently the only candidate in this area, due to its magnificent features as simplicity and fault tolerance, as well as the huge success and adoption of its implementation, namely Hadoop. Hadoop was developed by Yahoo!, and it has been widely used by both industry and academia.

Lots of research efforts have been done on improving the

performance of MapReduce-based application, implementing MapReduce in different environments [11, 14] and making MapReduce suitable for performing wider range of application such as scientific applications [15] or computing intensive application.

Due to the simplicity of the MapReduce paradigm, it is easy to write MapReduce program, but it is still hard and cumbersome to debug it and get a clear understanding beneath the veil. For example, launching a typical MapReduce job will involve a huge number of distributed processes and threads whose number equals to approximately two times of the participated machines plus each sub-process born out of map or reduce task in slave nodes, which makes it hard to monitor and analyze. Recently, many universities and enterprises have established educational courses and tutorials to help people understand the MapReduce algorithms. Mochi system is introduced to visualize the log files [4] and a universal tracing framework *X-Trace* is used to trace Hadoop workflow [5].

The objective of this research is to design and develop a user-friendly system with highly interactive user interface and visualize what is happening beneath Hadoop in real-time to multiple tracing clients. Our system has been motivated by following needs: accelerating the process of locating the bottlenecks and testifying the latest optimization method's efficiency under different circumstances, such as virtual clusters in CLOUDLET [11, 12]. In this paper, we present the design and implementation of a Hadoop's tracing framework MR-Scope, which is used to visualize Hadoop in a real-time manner, shorten the time between optimization thoughts and final efficiency report. MR-Scope is written in Java and is now seamlessly integrated into Hadoop as a module. We borrow the Hadoop's original RPC module and change a little bit to satisfy our needs, more details are given in section 4. MR-Scope visualizes Hadoop in three different but inter-connected perspectives: *Hadoop Distributed File System* (HDFS) perspective, MapReduce task scheduling perspective and MapReduce distributed perspective. Details are presented in section 4.

In summary, the main contributions of our work are:

- We are the first in visualizing the HDFS, capturing the detail information of the distribution of the filesystem blocks and their replicas in Hadoop cluster.
- We provide dynamic, on fly, sight on the running tasks including their progress and hosted nodes.
- We implement our MR-Scope visualization system in Hadoop 0.1 and test the overhead of our implementation.

The rest of this paper is organized as follows: Section 2 provides the background knowledge related to this work including an overview of the MapReduce model and Hadoop. Section 3

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

HPDC'10, June 20–25, 2010, Chicago, Illinois, USA.

Copyright 2010 ACM 978-1-60558-942-8/10/06 ...\$10.00.

discusses some related works. In section 4, we introduce the design and the implementation of the MR-Scope system. The evaluation of the MR-Scope overhead is presented in section 5. Section 6 discusses some open issues and the lessons learned from our implementation and experiments. Finally, we conclude the paper and propose our future work in section 7.

## 2. BACKGROUND

### 2.1 MapReduce Programming Model

MapReduce [1] is a programming model for data intensive computing inspired by the functional programming. It is represented in two functions: map and reduce. This model allows programmers to easily design parallel and distributed applications simply by writing Map/Reduce components, leave complicated issues such as parallelization, concurrency control and fault-tolerance to MapReduce runtime.

### 2.2 Hadoop

Hadoop [2] is a java open source implementation of MapReduce sponsored by Yahoo!. The Hadoop project is a collection of various subprojects for reliable, scalable distributed computing. The two fundamental subprojects are HDFS and Hadoop MapReduce framework.

HDFS is a distributed file system that provides high throughput access to application data. It is inspired by the GFS [3]. HDFS has master/slave architecture. The master server, called NameNode, splits files into blocks and distributes them across the cluster with replication for fault tolerance. It holds all metadata information about stored files. The HDFS slaves, the actual store of the data blocks called DataNodes, serve read/write requests from clients and execute replication tasks as directed by the Namenode.

Hadoop MapReduce framework is software package for processing large data sets on clusters [4]. It runs on top of HDFS. Thus data processing is co-located with data storage. It also has master/slave architecture. The master, called JobTracker (JT), is responsible of: (a) querying the Namenode for the block locations, (b) considering the information retrieved by the Namenode, JT schedule the tasks on the slaves, called TaskTrackers (TT), and (c) monitoring the successes and failures of the tasks.

## 3. RELATED WORK

Ignoring the implementation details, tracing could be roughly divided into static and dynamic methods which has their own advantages and disadvantages. In static method, Mochi [4] extracts and visualizes information about MapReduce programs at the MR-level abstraction, based on Hadoop's system log. It could give user trace figure correlates Hadoop's behavior in space, time and volume. Artemis [13] is a modular application designed for analyzing and troubleshooting the performance of large clusters running datacenter services. Artemis collects and filters the distributed logs and then stores them into database according to application specific schema.

In dynamic method, X-Trace [5, 6] is a universal tracing framework designed for wide range of internet protocols.

MR-Scope is the first tracing tool which adopts the dynamic method and is designed specifically for MapReduce. After MR-Scope starts, it could automatically retrieve Hadoop's current status, and then waiting for the tracing information sent back from Hadoop and then it is exactly synchronized with the running

Hadoop. A real-time visualization tool means more probability to locate other hidden potential bottlenecks because you can watch the visualization and dig your own interested information simultaneously. MR-Scope has a smaller basic tracing unit. Users could get and understand more. The implementation details will be discussed in section 4.

Unlike ParaGraph [16] or other MPI performance visualizers which take good advantage of the profiling interface of MPI defined in MPI standard [17], MR-Scope defines the tracing protocol by itself because there is no similar profiling mechanism in MapReduce.

## 4. DESIGN AND IMPLEMENTATION

### 4.1 MR-Scope Architecture

The framework of MR-Scope is composed of three layers: Hadoop Layer (Tracing Layer), RPC Layer (Communication Layer) and Client Layer as illustrated in Figure 1. MR-Scope users stay in Client Layer, and anybody who wants to add his own observation point should do some modification across these three layers. Although without official MapReduce profiling interface, this design could also bring convenience to extension.

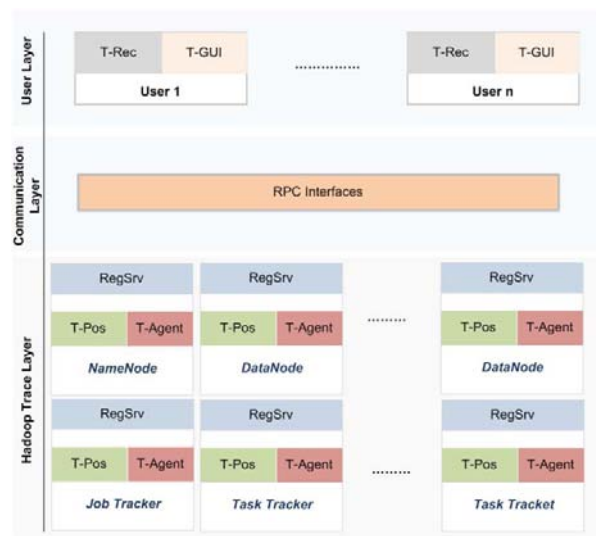


Figure 1. MR-Scope Design

Figure 1 shows a clear hierarchical relationship in different layers.

**Hadoop layer.** The Hadoop layer has three new modules besides original Hadoop implementation.

1. *Registration Server (RegSrv).* This module starts at the same time with its host process such as Hadoop system daemon process. Then *RegSrv* enters listening mode and accept users' register/unregister request. This module is a basis for *T-Agent*.
2. *Trace Agent (T-Agent).* This module is used as a *probe* or a *transmitter*, *T-Agent* transmits trace information to graph workstations which are on the *RegSrv*'s list.
3. *Trace Positive (T-Pos).* This module answers requests from users, typically after this user just starts and during the user's interaction with GUI.

**RPC interfaces.** A collection of protocols used to specify how client layer and Hadoop layer communicates. To make our system as a coherent module in Hadoop and avoid the troubles brought by making up an application protocol in socket communications, we change the Hadoop’s internal RPC mechanism to a non-block way to deal with irregular trace receivers, so death of trace receivers will not bring bad influence. Table 1 lists the complete protocols used in MR-Scope.

**Trace user layer.** This layer has two modules:

1. *Trace Receiver (T-Rec).* This module collects the distributed trace information sent back from multiple Trace Agents in memory and hands them to *T-GUI*.
2. *Trace GUI (T-GUI).* Visualize according to *T-Rec* in real-time.

*Observation point (OP)* is a special position in Hadoop’s source code, where usually a significant status change is taking place. Seeking precise OP is an essential and most time consuming step in constructing MR-Scope. However, the OP mechanism also brings lots of flexibility in extending MR-Scope, a typical extension to MR-Scope involves the following three steps: locating the OP’s position; defining it in Trace protocols; sending it by *T-Agent* and receive it by *T-Rec*.

The upper part of Figure 2 takes one classical OP (heartbeat) for example to explain how this tracing system works. Heartbeats are indispensable components in most of the distributed systems, slave machines demonstrate their coming and existence through sending heartbeat information to master machine, and master machine detects slave machines’ failure or abnormality by losing heartbeat information for a certain amount of time.

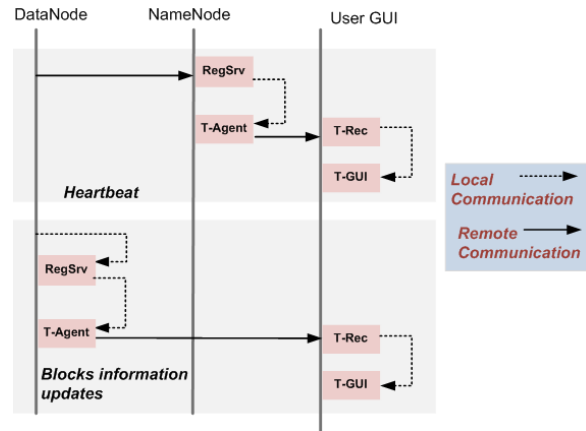
The lower part of Figure 2 shows that instead of transmitting the trace information to Namenode first, Datanode could directly notify the Trace receiver that blocks status has been changed (such as blocks’ quantity, blocks’ validity) and needs immediate update. This demo explains that every system process could act as a *trace source* if a *RegSrv* and *T-Agent* are injected into it.

## 4.2 MR-Scope Implementation

### 4.2.1 HDFS perspective

The topology of HDFS is like a tree. Namenode is the root and Datanodes are the leaves. Namenode has recorded all the meta-information needed in a file system. Datanode gets and executes the commands from the Namenode, furthermore, Datanodes report to Namenode what data blocks it has stored periodically.

Each time a DFS client wants to do some basic file system operations, instead of fetching data from a certain node, it finds out how many blocks this file is split into and in which Datanode they reside, and then it gets blocks’ data from their corresponding Datanodes one at a time until the operation is finished.



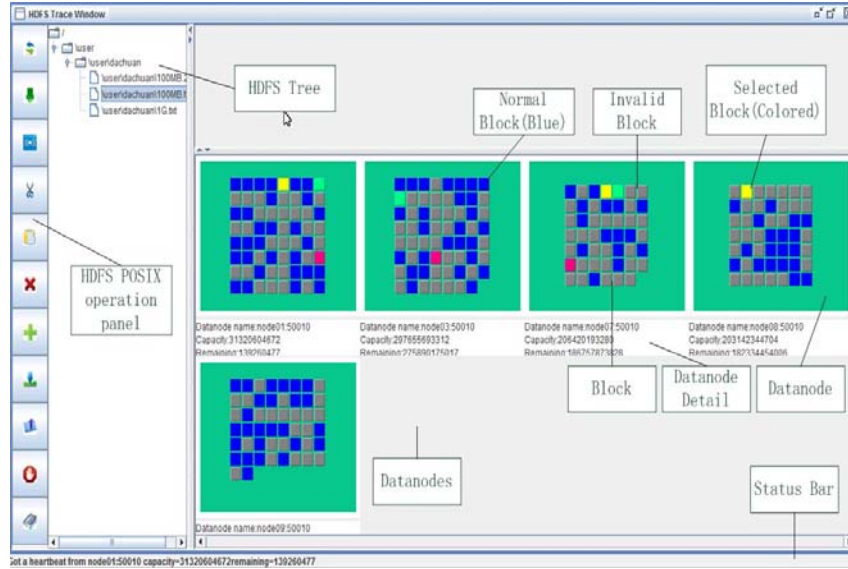
**Figure 2.** Use heartbeat and blocks information update as examples of observation point to demonstrate the two ways MR-Scope could work

Figure 3 is a screenshot of HDFS perspective. This display uses a little square area to represent a *Block* in order to give users a clear and an intuitive sense of HDFS. This display allows user to click on one *Block* to see its content, and when user’s mouse moves on one *Block*, some information will pop out, such as which file this *Block* belongs to, what this *Block*’s start position and length is. The left panel provides users POSIX-like file system operations choices, from up to bottom are *refresh*, *expand*, *copy*, *cut*, *paste*, *delete*, *mkdir*, *download*, *upload*, *redistribute*, and *cluster report*.

HDFS perspective gives user an unambiguous picture of HDFS, uses different colors to distinguish different data blocks, so the user could see the health condition of HDFS, how a big file is split apart and where the replicated blocks are located. User can see each block’s content and the blocks’ number growing dynamically when a file create operation under execution. Besides, HDFS perspective also acts as supportive module to MapReduce task scheduling perspective because the user can see every map task’s input block here, only by clicking the map task in MapReduce task scheduling perspective (See 4.2.2 for details). In HDFS perspective, the smallest tracing unit is *Block*.

**Table 1: MR-Scope’s protocols (include the protocols used in Hadoop system.)**

Protocol name	Used by	Implemented by	Type	Method
DatanodeProtocol	Datanode	Namenode	System Protocol	RPC
InterTrackerProtocol	TaskTracker	JobTracker	System Protocol	RPC
JobSubmissionProtocol	JobClient	JobTracker	System Protocol	RPC
MapOutputProtocol	Reduce Task	TaskTracker	System Protocol	RPC
TaskUmbilicalProtocol	Map/Reduce Task	TaskTracker	System Protocol	RPC
TracePassiveProtocol	Trace Agent	Trace Receiver	Trace protocol	RPC
T-PosJobTrackerProtocol	Trace user	JobTracker	Trace protocol	RPC
T-PosNamenodeProtocol	Trace user	Namenode	Trace protocol	RPC
RegisterProtocol	Trace Receiver	RegSrv	Trace Protocol	Socket
TraceVisualProtocol	Trace Receiver	Trace GUI	Trace Protocol	Local



**Figure 3: Screenshot of HDFS perspective. In this screenshot, HDFS has two 86MB file and a 1GB file. Three color blocks means 86MB file is comprised of three blocks (nearly 32MB each) and three yellow color blocks means this block has three replications.**

HDFS perspective is a great help in testing HDFS performance. Users could get an intuitive sense of issues like load balance, read/write performance, the amount of expired blocks, Datanode's capacity, etc. Moreover, users could use this perspective to test the feasibility of his/her hypothesis in improving HDFS's performance, such as replacing the old load-balancing algorithm.

HDFS perspective has nine OPs, includes Datanode failure event, datanode heartbeat event, blocks information update event, blocks expired event, etc. The complete list is:

1. *Goheartbeat*. Invoked when a Datanode sends its heartbeat to Namenode.
2. *GotNewHeartbeat*. Invoked when a Datanode has just joined in HDFS.
3. *UpdatePendingCreates*. Invoked when a new file's blocks have just been completed.
4. *DeletePendingCreates*. Invoked when a new file's creation has been abandoned.
5. *UpdateRecentInvalidateSets*. Invoked when some blocks on a Datanode expired.
6. *DeleteRecentInvalidateSets*. Invoked when expired blocks have been deleted on a Datanode.
7. *UpdateNeededReplications*. Invoked when blocks which need replication have been updated.
8. *UpdateDatanodeBlocks*. Invoked when blocks on a Datanode have been updated.
9. *RemoveDataNode*. Invoked when a Datanode left.

#### 4.2.2 MapReduce task scheduling perspective

In MapReduce task scheduling perspective, the smallest tracing unit is *Map Task* or *Reduce Task*, such as the current progress of

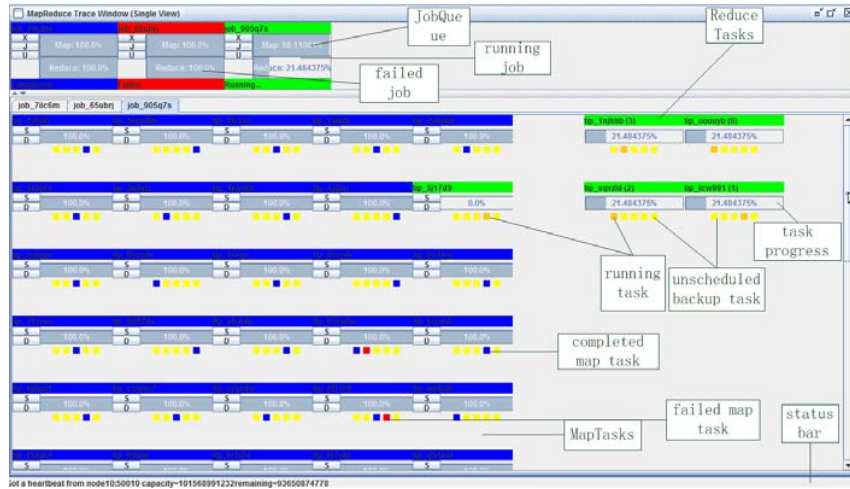
each task, it also gives people how a *Map* or *Reduce* task is scheduled.

Table 2 shows three possible scheduling options: cache-hit scheduling (which means this map task's input split is exactly located in the same machine with TaskTracker, only map task could be scheduled in a cache-hit way), standard scheduling (for map task, this means the opposite of cache-hit, for reduce task, this is the default schedule way) and speculative scheduling [9] (which means one task is a slow straggler and then a speculative task is scheduled to compete with it).

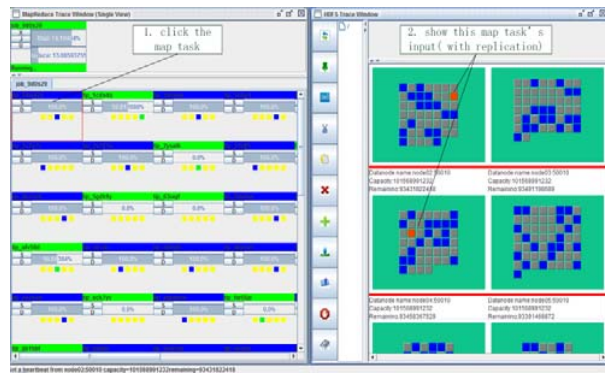
**Table 2: Three options map/reduce tasks expected to be scheduled**

	Map Task	Reduce Task
<i>Cache-hit</i>	Scheduled to a node which holds its input	-
<i>Standard</i>	Scheduled to a node which doesn't hold its input	Scheduled to a node in a default way.
<i>Speculative</i>	Scheduled to a node because a predecessor map task who has the same input is a straggler	Scheduled to a node because a predecessor reduce task who has the same input is a straggler

Figure 4 is a screenshot of MapReduce task scheduling system. This display uses several little square areas to represent a *scheduling pool* in Map or Reduce task. Each task has five opportunities to run, so each task has five little squares beneath. This display allows users to click on buttons beside *task panel* to get more information, click the button on *map task panel* this display will lead us to HDFS perspective to see this task's input, as illustrated in Figure 5. When user's mouse moves on these squares, the start time and host node's name will appear, and when user clicks buttons beside a Job, he/she could also jump to browser to watch the static report provided by Hadoop.



**Figure 4. MapReduce task scheduling perspective.** Test case: JobName: wordcount, Input Data Size: 2GB, Reducers: 4, Slave Nodes: 8. As we can see from the figure, each task has been granted five opportunities to run (could fail at most four times) and two map tasks have failed once (small red) and been re-scheduled to another slave.



**Figure 5: A Map Task's input block in HDFS perspective**

This perspective could be used to demonstrate the effectiveness of task scheduling policy and data split policy, a perfect policy is that map tasks' slots are always exhausted and map tasks always finish at the same time because input has the same size. This perspective could also be used to show the health and availability of working nodes. If tasks frequently fail at this node, the administrator should consider shut it down.

MapReduce Task scheduling perspective has 14 OPs, includes job status change event, task status change event, tasktracker leaving event, task scheduling event, etc. The complete list is:

1. *EmitHeartbeat*. Invoked when TaskTracker sends a heartbeat to JobTracker.
2. *NeedsUpdate\_jobInitQueue*. Invoked when a new Job has been submitted.
3. *LostTaskTracker*. Invoked when a TaskTracker fails.
4. *Job\_FileSplits\_ready*. Invoked when a job's input has been divided.

5. *Job\_JobStatus\_update*. Invoked when a job's status has been updated.
6. *New\_mapTIP\_inited*. Invoked when a new map TIP has been started.
7. *New\_reduceTIP\_inited*. Invoked when a new reduce TIP has been started.
8. *Job\_TIPs\_inited*. Invoked when a job's all TIPs have been started.
9. *TIP\_progress\_update*. Invoked when a TIP's progress has been updated.
10. *Taskstatus\_update*. Invoked when a task's status has been updated.
11. *TIP\_subTask\_failed*. Invoked when a task's failure has been reported.
12. *TIP\_task\_completed*. Invoked when a task has been completed.
13. *TIP\_killed*. Invoked when a TIP has been killed.
14. *TIP\_task\_scheduled*. Invoked when a task has been scheduled to TaskTracker.

## 5. PERFORMANCE EVALUATION

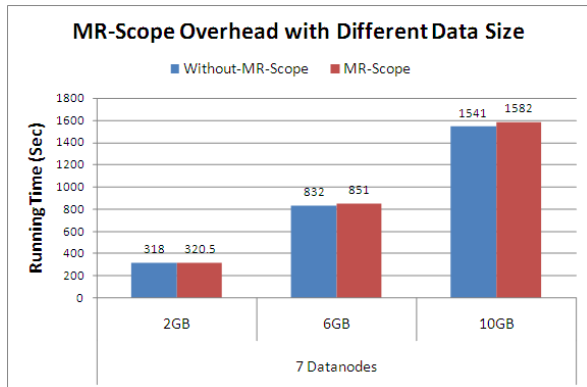
Our experiments mainly focus on the overhead brought by the trace system because it has made changes to the Hadoop source code and some additional trace data needs to be transferred in network. The evaluation environment consists of eight nodes cluster (one node for master-node, and the others for slave-nodes). Each node in the cluster is equipped with two quad-core 2.33GHz Xeon processors, 8GB of memory and 1TB of disk, runs RHEL5 with kernel 2.6.22, and is connected with 1 Gigabit Ethernet.

In our experiments, we set *mapred.reduce.tasks* as 4 (every job has four Reducers); and set *dfs.replication* as 2 (every block has one replication besides itself). The benchmark is *WordCount*. We compare the performance between native Hadoop system with no

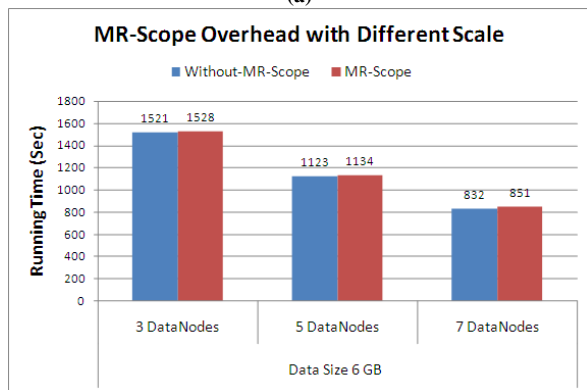
trace clients and our trace-Hadoop with 5 trace clients under different datasizes and different slaves' number.

We took data input size and slaves number as two most significant factor which could have great influence on MR-Scope's overhead, we fix each one and grow another (slaves number is fixed in first graph, and data input size is fixed in the second graph).

We could conclude from Figure 6, the new kernel produces at most 3% overheads (the severest overhead comes under 7 Datanodes and 10GB input, about 2.6%). The overheads will slightly increase as the input data grows but will not exceed 4%.



(a)



(b)

Figure 6: MR-Scope Overhead: (a) with different data (b) with different cluster scale

## 6. LESSON LEARNED AND OPEN ISSUES

MapReduce framework embeds error-handling, task division and task scheduling module into its runtime environment. This brings programmers lots of conveniences. However, a typical high-efficient MapReduce Job involves too many subtle configuration properties adjustment which needs knowledge about the internal workflow of MapReduce.

Some difficulties met during the development of MR-Scope are common to other real-time monitoring-related system: the sequence of trace messages and loss of messages. In a multi-threaded programming environment, especially in a distributed cluster, the trace messages flood does not likely reach the trace clients in the same sequence. So if an incremental message schema

(which means the content of one message depends on previous message) is adopted for real-time tracing system, it will not be right if this problem happened. Absolute message schema is chosen for our system instead. The loss of message is another big problem, so in our system we send some message twice if this message has a vital influence on following operation.

One of the challenges faced by real-time tracing system is working quietly in the background without compromising the host system's performance, so whether we should choose some frequent called code with important status change as observation point remains an open issue.

## 7. CONCLUSION AND FUTURE WORK

In this paper, we develop a real-time tracing system for MapReduce - to give clearer image on the insight process of MapReduce. In addition to visualize the on-going progress of each running task, MR-Scope displays the distribution of the filesystem blocks and their replicas as well as views the content of the blocks. We implement MR-Scope in Hadoop-0.1. Experimental results show that when running wordcount benchmark, MR-Scope add relatively small overhead of 4%.

The MapReduce distributed perspective is our on-going work. We view MapReduce task scheduling perspective as scarce of fine granularity; lots of other hidden details should be uncovered to show users a clear image. Our plan is digging into MapReduce more deeply, to trace every sub-phase in each task. For instance, 1) show every map task's output files' size in order to achieve better policies for data distribution during the shuffle phase, as well as to judge how many data have been decreased using combine function in map task; and 2) show the time period in each phase of reduce task. With all these observation points, we could get where a <key,value> pair comes and where it goes, and could see a more clear image when some optimization method is adopted.

Furthermore, we would like to add another feature in our system: deploying Hadoop in virtual clusters, extending our prototype system to latest version of Hadoop package and test our system in large scale cluster.

## 8. ACKNOWLEDGMENTS

This work is supported by National 973 Key Basic Research Program under grant No. 2007CB310900, Information Technology Foundation of MOE and Intel under grant MOE-INTEL-09-03, National High-Tech R&D Plan of China under grant 2006AA01A115, China Next Generation Internet Project under grant CNGI2008-109, Key Project in the National Science & Technology Pillar Program of China under grant No.2008BAH29B00.

## 9. REFERENCES

- [1] Dean, J. and Ghemawat, S. MapReduce: Simplified Data Processing on Large Clusters. In *Proceedings of OSDI*. 2004, 137-150.
- [2] Hadoop: <http://hadoop.apache.org/2010>.
- [3] Ghemawat, S., Gobiuff, H., and Leung, S. The Google file system. In *Proceedings of SOSP*. 2003, 29-43.
- [4] Tan, J., Pan, X., Kavulya, S., Gandhi, R., and Narasimhan, P. Mochi: Visual Log-Analysis Based Tools for Debugging Hadoop. *CMU-PDL-09-103*, Technical Report, May 2009.

- [5] Fonseca, R., Porter, G., Katz, R. H., Shenker, S., and Stoica, I. X-Trace: A Pervasive Network Tracing Framework. In *Proceedings of NSDI*. 2007.
- [6] Konwinski, A., Zaharia, M., Katz, R., and Stoica I. X-Tracing Hadoop. *Presentation in Hadoop Summit*, Mar 2008.
- [7] Massie, M. L., Chun, B. N., and Culler, D. E. The ganglia distributed monitoring system: design, implementation, and experience. In *Parallel Computing*. Vol.30, No.7, 2004, 817-840.
- [8] Quiroz, A., Gnanasambandam, N., Parashar, M., and Sharma, N. Robust clustering analysis for the management of self-monitoring distributed systems. In *Proceedings of Cluster Computing*. 2009, 73-85.
- [9] Zaharia, M., Konwinski, A., Joseph, A. D., Katz, R. H., and Stoica, I. Improving MapReduce Performance in Heterogeneous Environments. In *Proceedings of OSDI*. 2008, 29-42.
- [10] Boulon, J., Konwinski, A., Qi, R., Rabkin, A., Yang, E., and Yang, M. Chukwa: A large-scale monitoring system. In *Proceedings of Cloud Computing and Its Applications*, 2008.
- [11] Ibrahim, S., Jin, H., Cheng, B., Cao, H., Wu, S., and Qi, L. CLOUDLET: towards mapreduce implementation on virtual machines. In *Proceedings of HPDC*. 2009, 65-66.
- [12] Ibrahim, S., Jin, H., Lu, L., Qi, L., Wu, S., and Shi, X. Evaluating MapReduce on Virtual Machines: The Hadoop Case. In *Proceedings of CloudCom*. 2009, 519-528.
- [13] Cretu-Ciocarlie, G. F., Budiu, M., and Goldszmidt, M. Hunting for Problems with Artemis. In *Proceedings of WASL*. 2008.
- [14] He, B. S., Fang, W., Luo, Q., Govindaraju, N. K., and Wang, T. Mars: a MapReduce framework on graphics processors. In *Proceedings of PACT*. 2008, 260-269.
- [15] Chen, S, and Schlosser, S. W. Map-Reduce Meets Wider Varieties of Applications. *IRP-TR-08-05*, Technical Report, Intel Research Pittsburgh, May 2008.
- [16] ParaGraph User Guide. ParaGraph: A Performance Visualization Tool for MPI <http://www.csar.illinois.edu/software/paragraph/>
- [17] MPI standard: <http://www.mpi-forum.org/docs/mpi-11-html/mpi-report.html> 2010.