ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 3, Issue 8, August 2015

Design and Evaluation of Hierarchical Merge with RDT Algorithm for Hadoop Acceleration

Sonali R. Jagtap, Vaibhav Dhore

M.E, Dept. of Computer Engineering, R. M. D. Sinhgad College of Engineering, Pune, India

Assistant Professor, Dept. of Computer Engineering, R. M. D. Sinhgad College of Engineering, Pune, India

ABSTRACT: Map Reduce is programming model and it processes large scale data in distributed applications due to its efficiency, simplicity and ease of use. Hadoop is an open source implementation of the Map Reduce programming model for cloud computing. Thought Hadoop is giving best performance but still it tackle number of issues to gain that performance . The issues faces by Hadoop are (1) A serialization barrier that delays the reduce phase, repetitive merges, and disk accesses (2) The lack of portability to different interconnects. In this paper, propose a distributed implementation of Hierarchical Merge using RDT algorithm using Map Reduce computing model, and deploy it on a Hadoop cluster. It resolve memory scalability problem and increases performance of Map reduce. It also speed up the delivery of data from MapTasks to Reduce Tasks. While their work reduces job completion time and improves system utilization.

KEYWORDS: Hadoop; MapReduce; virtual shuffling; hierarchical merge; Hadoop acceleration.

I. Introduction

Map-Reduce [1] has emerged as a popular and easy-to-use programming model for cloud computing . It has been used by numerous organizations to process explosive quantity of data, perform large number of computation, and fetch critical knowledge for business intelligence. Hadoop [2] is an open source implementation of MapReduce, currently maintained by the Apache Foundation, and supported by leading Information technology companies such as Facebook and Yahoo!.Hadoop is a implementation of MapReduce framework with two types of components: a Job Tracker and many Task-Trackers. The Job Tracker commands Task Trackers (a.k.a.slaves) to process data in parallel through two main functions: map and reduce. In this process, the JobTracker is in charge of scheduling map tasks (MapTasks) and reduce tasks (ReduceTasks) to Task Trackers.

It also monitors their growth, collects runtime execution result, and handles possible errors and faults through task reexecution. Between the two phases, the intermediate output from all finished MapTasks need to be fetch by reduce task. Globally, this leads to the shuffling of intermediate data (in segments) from all MapTasks to all ReduceTasks. For many data intensive MapReduce programs, data shuffling can cause to a significant number of operations on disk, contending for the limited I/O bandwidth. This severe problem of disk I/O contention in MapReduce programs, which entails further research on efficient data shuffling and merging algorithms.Fig. 1 shows overview of data processing in Hadoop.

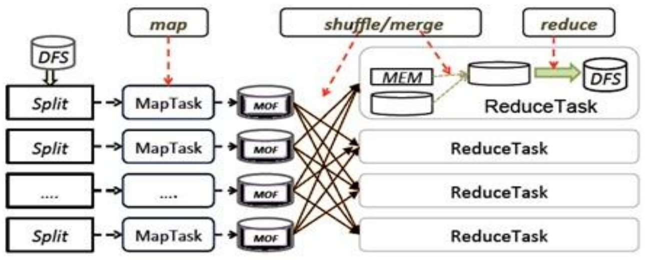


Fig.4. Data Processing In Hadoop MapReduce Framework

Improving the performance of MapReduce recently.Originally, the Hadoop scheduler assumed that all nodes in a cluster were homogeneous and made progress with the same speed. Focuses on fast data output from MapTasks and changes Hadoop fault handling mechanism. The major issues faces by hadoop are Serialization barrier and continuous merging of data and disk access.

A. *Serialization barrier in Hadoop* :

Hadoop is easy to use data processing framework.When large number of task is given for processing a pool of concurrent MapTasks starts the map function on the first set of data splits. When Map Output Files (MOFs) are generated from these splits, a pool of ReduceTasks starts to fetch partitions from these MOFs. At each ReduceTask, when the number of segments is larger than a threshold, or when their total data size is more than a memory threshold, the smallest segments are merged. For the correctness of the MapReduce programming model, it is necessary to ensure that the reduce phase does not start until the map phase is done for all data splits. However, the pipeline, as shown in Fig.2, contains an implicit serialization. At each ReduceTask, only until all its segments are available and merged, will the reduce phase start to process data segments via the reduce function. This essentially enforce a serialization between the shuffle/ merge phase and the reduce phase. When there are many segments to process (which is often the case), it takes a significant amount of time for a ReduceTask to shuffle and merge them. As a result, the reduce phase will be significantly delayed and this can increase the total execution time.

B. *Continuous Merging and disk access:*

Hadoop ReduceTasks merge data segments when the number of segments or their total size goes over a threshold.
A newly merged segment has to be spilled to local disks due to memory pressure. However, the current merge algorithm in Hadoop often leads to repetitive merges, thus extra disk accesses. Fig. 3 shows a common sequence of merge operations in Hadoop.When more segments arrive, the threshold is reached again. It is then necessary to merge another set of segments.This again causes additional disk access, let alone the need to read segments back if they have been stored on local disks. As even more segments arrive, a previously merged segment will be grouped into another set and merged again. Furthermore, any segment merged from a subset of segments eventually needs to be merged for final results. Altogether, this means repetitive merges and disk access, causing degraded performance for Hadoop. Therefore, an alternative merge algorithm is critical for Hadoop to mitigate the impact of repetitive merges and extra disk accesses.

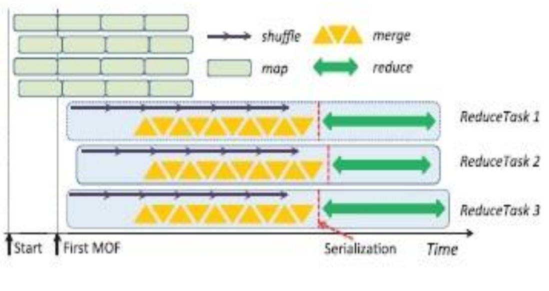


Fig.2. Serialization barrier in Hadoop

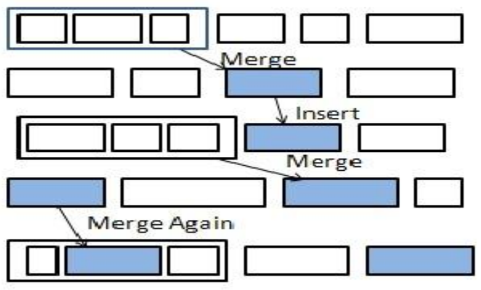


fig.3. Continuous Merging and disk access

II. RELATED WORK

The MapReduce[1] is programming model has been successfully used. This model is very easily used by the programmers not having experience with parallelization and distributed systems, since it handles issues in distributed system such as parallelization, fault-tolerance, locality optimization, and load balancing. Hadoop Mapreduce[12] is the best tool available for processing data and its distributed, column-oriented database, HBase which uses HDFS for its underlying storage, and support provides more efficiency to the system.But still hadoop having some data shuffling problem and data shuffling can lead to a significant number of disk operations, contending for the limited I/O bandwidth. In network-levitated merging algorithm [3] the idea is to leave data on remote disks until it is time to merge the intended data records. In network levitated algorithm, number of remote segments S1, S2,..., Sn are to be fetched from local disks and then merged. the alternative way of accessing them to local disks,network levitated algorithm only fetches a small header from each segment. Each header is made up of partition length, offset, and the first pair of < key;

val >.These < key; val > pairs are enough to build a priority queue (PQ) to manage these segments. After the first < key; val >pair,many records can be fetched as according to the available memory. Because it access only a tiny amount of data per segment, this algorithm does not store or merge segments onto local disks. When the number of segments is over a threshold without merging them, we keep building up the Priority Queue until all headers arrive and are merge. When the building of PQ has been completed, the merge phase starts its execution. The leading < key; val > pair will be the beginning point of merge operations for individual segments, i.e., the merge point.The design and architecture of Hadoops MapReduce framework propose by author Mrigank mridual, Kumar N[4].In hadoop analysis is based on data processing. So bigdata is the new buzz word and Hadoop Mapreduce is the best tool available for processing data and its distributed, column-oriented database, HBase which uses HDFS for its underlying storage, and support provides more efficiency to the system.But still hadoop having some data shuffling problem and data shuffling can lead to a significant number of disk operations, contending for the limited I/O bandwidth.

MapReduce is a programming model for large-scale arbitrary data processing. To fully take advantage of the multicore and multiprocessor systems, Ranger et al. [4] designed Phoenix,a MapReduce implementation for shared-memory systems. In Phoenix, users only need to write simple parallel code without considering the complexity of thread creation, dynamic task scheduling, data partitioning, and fault tolerance across processor nodes. Kaashoek et al. [5] then designed a new MapReduce library with a compromise data structure, which outperforms its simpler peers, including Phoenix. Tiled-MapReduce designed by Chen et al. [6] further improves the Phoenix by leveraging the tiling strategy that is commonly used in complier community. It divides a large MapReduce job into multiple discrete subjobs and extends the Reduce phase to process partial map output. By meticulously reusing memory and threads, Tiled-MapReduce achieves considerable speedup over Phoenix. But our work is completely different from these works in three aspects.

III. Proposed System

A. *System Architecture of Hadoop:*

Figure 4. shows architecture of proposed system.The group of Task is given to the system,this group is splitting and given for the mapping.Map phase take the task and generate intermediate file this is based on RDT.The intermediate file get shuffle and merged and reducer reduces the result. All this phases generate tree so that performance of system get increases and because we create tree for each phase memory require is less.

B. *Algorithm For Random decision tree:*

C4.5 is a standard algorithm for inducing classification rules in the form of decision tree. As an extension of ID3 , Fig.4. System Architecture of System the default criteria of choosing splitting attributes in C4.5 is information gain ratio. Instead of using information gain as that in ID3, information gain ratio avoids the bias of selecting attributes with many values.

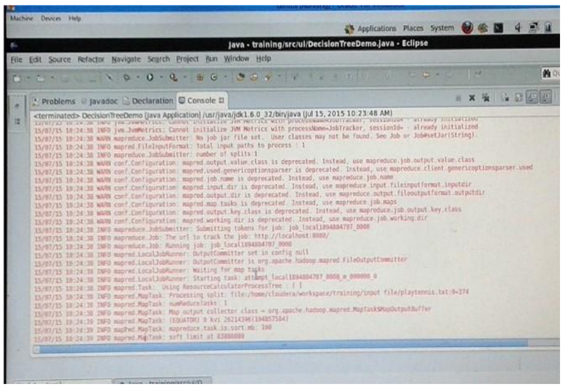
```
Input: Dataset T, Attribute S;
Output: Decision Tree tree;
  If  T is Null then
      return Fail
  end If  s=1 then
      return Tree as a single Node
  end If
  set Tree=Empty
  for a  subset S do
      setInfo(a; T) = 0 and  SplitInfo(a; T) = 0
      Compute Entropy(a);
      for v  belongs values(a; T) do
        Set Ta;v as a subset of T as attribute a=v
        Info(a,T)+=Ta;v=TaEntropy(av)
       splitInfo(a,T)=  -(Ta;v=Ta)Log(Ta;v=Ta)
    end for
```

```
      Gain(a,T)=Entropy(a)-Info(a,T)
      GainRatio(a,T)=Gain(a,T)/SplitInfo(a,T)
 end for
 set a_best = argmaxGainRatio(a; T) attach  a_best into Tree
 for v  values(a_best,T) do
     call again
end For
Return Tree
```

IV. Simulation Results

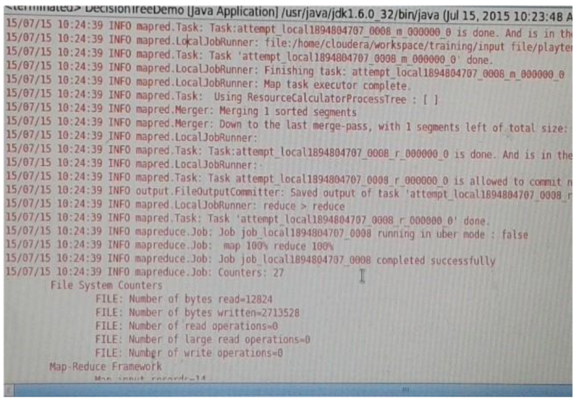
The system conduct our experiment on 1 master node and 1 slave node having 5 GB RAM. This node run ubantu operating system name centos cloudera having 1 processor and 1 TB hard disk.Weather forecasting dataset.we require hadoop framework and apache framework with eclipse. The forecasting dataset is given as a input to the system. The Figure 5. shows the final execution of system.fig 6. Shows memory require for the file system which include no. of bytes read ,written,operation on file system etc.fig 7 display final result in which number of map task ,reduce task,memory required ,number of nodes ,entropy is shown.

When total execution of system is completed the RDT is generated for final result based on dataset shown in fig.8.For intermediate files also RDT is generated shown in fig. 9.In this experiment 8 homogeneous clusters are generated from weather forecasting dataset.Number of Clusters are generated according to size of dataset at runtime.Fig. 10 shows the Fig. 7. Final Result cluster information ,intermediate file , Source file and output file.


Machine Devices Help
Applications Places System
Java - training/src/ui/DecisionTreeDemo.java - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
Problems Javadoc Declaration Console

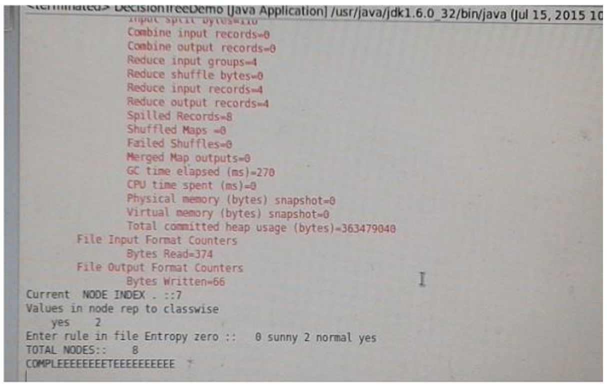
```
<terminated> DecisionTreeDemo [Java Application] /usr/java/jdk1.6.0_32/bin/java (Jul 15, 2015 10:23:48 AM)
15/07/15 10:24:38 INFO jvm.JvmMetrics: Cannot initialize JVM Metrics with processName=JobTracker, sessionId= - already initialized
15/07/15 10:24:38 WARN mapreduce.JobSubmitter: No job jar file set.  User classes may not be found. See Job or Job#setJar(String).
15/07/15 10:24:38 INFO mapred.FileInputFormat: Total input paths to process : 1
15/07/15 10:24:38 INFO mapreduce.JobSubmitter: number of splits:1
15/07/15 10:24:38 WARN conf.Configuration: mapred.output.value.class is deprecated. Instead, use mapreduce.job.output.value.class
15/07/15 10:24:38 WARN conf.Configuration: mapred.used.genericoptionsparser is deprecated. Instead, use mapreduce.client.genericoptionsparser.used
15/07/15 10:24:38 WARN conf.Configuration: mapred.job.name is deprecated. Instead, use mapreduce.job.name
15/07/15 10:24:38 WARN conf.Configuration: mapred.input.dir is deprecated. Instead, use mapreduce.input.fileinputformat.inputdir
15/07/15 10:24:38 WARN conf.Configuration: mapred.output.dir is deprecated. Instead, use mapreduce.output.fileoutputformat.outputdir
15/07/15 10:24:38 WARN conf.Configuration: mapred.map.tasks is deprecated. Instead, use mapreduce.job.maps
15/07/15 10:24:38 WARN conf.Configuration: mapred.output.key.class is deprecated. Instead, use mapreduce.job.output.key.class
15/07/15 10:24:38 WARN conf.Configuration: mapred.working.dir is deprecated. Instead, use mapreduce.job.working.dir
15/07/15 10:24:38 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_local1894804707_0008
15/07/15 10:24:38 INFO mapreduce.Job: The url to track the job: http://localhost:8080/
15/07/15 10:24:38 INFO mapreduce.Job: Running job: job_local1894804707_0008
15/07/15 10:24:38 INFO mapred.LocalJobRunner: OutputCommitter set in config null
15/07/15 10:24:38 INFO mapred.LocalJobRunner: OutputCommitter is org.apache.hadoop.mapred.FileOutputCommitter
15/07/15 10:24:38 INFO mapred.LocalJobRunner: Waiting for map tasks
15/07/15 10:24:38 INFO mapred.LocalJobRunner: Starting task: attempt_local1894804707_0008_m_000000_0
15/07/15 10:24:38 INFO mapred.Task:  Using ResourceCalculatorProcessTree : [ ]
15/07/15 10:24:38 INFO mapred.MapTask: Processing split: file:/home/cloudera/workspace/training/input file/playtennis.txt:0+374
15/07/15 10:24:38 INFO mapred.MapTask: numReduceTasks: 1
15/07/15 10:24:38 INFO mapred.MapTask: Map output collector class = org.apache.hadoop.mapred.MapTask$MapOutputBuffer
15/07/15 10:24:39 INFO mapred.MapTask: (EQUATOR) 0 kvi 26214396(104857584)
15/07/15 10:24:39 INFO mapred.MapTask: mapreduce.task.io.sort.mb: 100
15/07/15 10:24:39 INFO mapred.MapTask: soft limit at 83886080
```

Fig.5.System Execution



```
<terminated> DecisionTreeDemo [Java Application] /usr/java/jdk1.6.0_32/bin/java (Jul 15, 2015 10:23:48 A
15/07/15 10:24:39 INFO mapred.Task: Task:attempt_local1894804707_0008_m_000000_0 is done. And is in th
15/07/15 10:24:39 INFO mapred.LocalJobRunner: file:/home/cloudera/workspace/training/input file/playte
15/07/15 10:24:39 INFO mapred.Task: Task 'attempt_local1894804707_0008_m_000000_0' done.
15/07/15 10:24:39 INFO mapred.LocalJobRunner: Finishing task: attempt_local1894804707_0008_m_000000_0
15/07/15 10:24:39 INFO mapred.LocalJobRunner: Map task executor complete.
15/07/15 10:24:39 INFO mapred.Task:  Using ResourceCalculatorProcessTree : [ ]
15/07/15 10:24:39 INFO mapred.Merger: Merging 1 sorted segments
15/07/15 10:24:39 INFO mapred.Merger: Down to the last merge-pass, with 1 segments left of total size:
15/07/15 10:24:39 INFO mapred.LocalJobRunner:
15/07/15 10:24:39 INFO mapred.Task: Task:attempt_local1894804707_0008_r_000000_0 is done. And is in the
15/07/15 10:24:39 INFO mapred.LocalJobRunner:
15/07/15 10:24:39 INFO mapred.Task: Task attempt_local1894804707_0008_r_000000_0 is allowed to commit n
15/07/15 10:24:39 INFO output.FileOutputCommitter: Saved output of task 'attempt_local1894804707_0008_r
15/07/15 10:24:39 INFO mapred.LocalJobRunner: reduce > reduce
15/07/15 10:24:39 INFO mapred.Task: Task 'attempt_local1894804707_0008_r_000000_0' done.
15/07/15 10:24:39 INFO mapreduce.Job: Job job_local1894804707_0008 running in uber mode : false
15/07/15 10:24:39 INFO mapreduce.Job:  map 100% reduce 100%
15/07/15 10:24:39 INFO mapreduce.Job: Job job_local1894804707_0008 completed successfully
15/07/15 10:24:39 INFO mapreduce.Job: Counters: 27
        File System Counters
                FILE: Number of bytes read=12824
                FILE: Number of bytes written=2713528
                FILE: Number of read operations=0
                FILE: Number of large read operations=0
                FILE: Number of write operations=0
        Map-Reduce Framework
```

Fig.6.Memory Required For System



```
<terminated> DecisionTreeDemo [Java Application] /usr/java/jdk1.6.0_32/bin/java (Jul 15, 2015 10
                Combine input records=0
                Combine output records=0
                Reduce input groups=4
                Reduce shuffle bytes=0
                Reduce input records=4
                Reduce output records=4
                Spilled Records=8
                Shuffled Maps =0
                Failed Shuffles=0
                Merged Map outputs=0
                GC time elapsed (ms)=270
                CPU time spent (ms)=0
                Physical memory (bytes) snapshot=0
                Virtual memory (bytes) snapshot=0
                Total committed heap usage (bytes)=363479040
        File Input Format Counters
                Bytes Read=374
        File Output Format Counters
                Bytes Written=66
Current  NODE INDEX . ::7
Values in node rep to classwise
    yes    2
Enter rule in file Entropy zero ::   0 sunny 2 normal yes
TOTAL NODES::    8
COMPLEEEEEEEETEEEEEEEEEE
```

Fig.7. Final Result

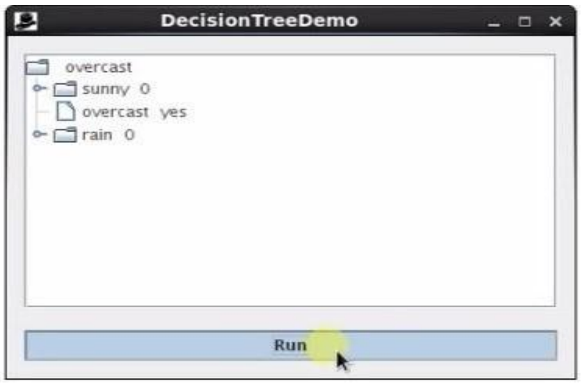


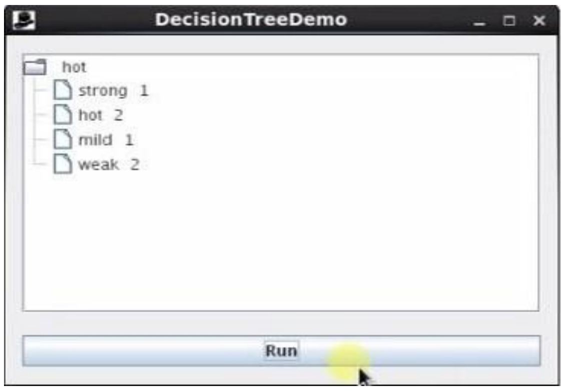
Fig.8.Final RDT ofSystem

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 3, Issue 8, August 2015



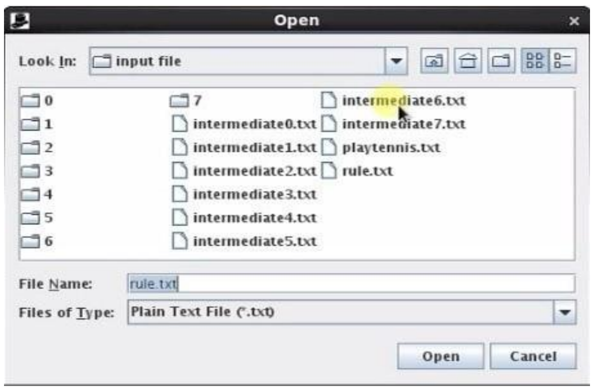


Fig.9. Intermediate file RDT

Fig.10. Clusters, Input, Intermediate, Output files

V. CONCLUSION AND FUTURE WORK

In this paper,we are describing Hierarchical merge with RDT algorithm to resolve Hadoop issues. the propose hierarchical merge with RDT algorithm that merges data without touching disks and designing a full Queue of shuffle, merge, and reduce phases for Reduce Tasks.Direct network channels between MapTasks and Reduce Tasks and speed up the delivery of data from MapTasks to Reduce Tasks. While their work reduces job completion time and improves system utilization.The structure of a random tree is constructed completely independent of the training data. The training data is scanned exactly once to update the statistics in multiple random trees. The memory requirement is to store multiple random trees and one data item at any given time. With the necessary number of random trees to ensure optimality, the memory requirement of the random tree algorithm is significantly less even if the very large training data could have been held entirely in main memory to compute the single best tree.

REFERENCES

1. J. Dean and S. Ghemawat, “MapReduce: Simplified Data Processing on Large Clusters,” Proc. Sixth Symp. Operating System Design and Implementation (OSDI ’04), pp. 137-150, Dec. 2004.
2. ApacheHadoop Project, http://hadoop.apache.org/, 2013.
3. Design and Evaluation of Network-Levitated Merge for Hadoop Acceleration Weikuan Yu, member, IEEE, Yandong Wang, and Xinyu Que
4. C. Ranger, R. Raghuraman, A. Penmetsa, G.R. Bradski, and C. Kozyrakis, “Evaluating MapReduce for Multi-Core and Multiprocessor Systems,” Proc. IEEE 13th Int’l Symp. High Performance Computer Architecture (HPCA ’07), pp. 13-24, 2007.
5. Y. Mao, R. Morris, and F. Kaashoek, “Optimizing MapReduce for Multicore Architectures,” Technical Report MIT-CSAIL-TR-2010- 020, Massachusetts Inst. of Technology, May 2010.
6. R. Chen, H. Chen, and B. Zang, “Tiled-MapReduce: Optimizing Resource Usages of Data-Parallel Applications on Multicore with Tiling,” Proc. 19th Int’l Conf. Parallel Architectures and Compilation Techniques (PACT ’10), pp. 523-534, 2010.
7. S. Babu, “Towards Automatic Optimization of MapReduce Programs,” Proc. First ACM Symp. Cloud Computing (SoCC ’10),pp. 137-142, 2010.
8. Globalarraystoolkit,http://www.emsl.pnl.gov/docs/global.
9. Report on experimental language X10, http://dist.codehaus.org/x10/documentation/languagespec/x10-170.pdf (2008).
10. LaSA: A Locality-aware Scheduling Algorithm for Hadoop-MapReduce Resource Assignment Tseng-Yi Chen, Hsin-Wen Wei, Ming-Feng Wei, Ying-Jie Chen, Tsan-sheng Hsu, Wei-Kuan Shih.