ISSN: 2278-0181
NCICN-2015 Conference Proceedings

Evaluating Cost Function of Micro Partitioning Strategies in Hadoop Cluster for Optimal Resource Provisioning

R. Sreedevi
Department of Computer Science and Engineering,
J.K.K. Nattraja College of Engineering and Technology,
Kumarapalayam, Tamilnadu, India

C. Vairavel
Department of Computer Science and Engineering,
J.K.K. Nattraja College of Engineering and Technology,
Kumarapalayam, Tamilnadu, India

***Abstract-* Due to the massive improvement in the usage of data's in the real world, it becomes more burdens to handle and process it effectively. The Map reduce is the one of the more developed technology which is used to handle and process the big data/largest tasks. Map reduce is sued to partition the task into sub partitions and map those partitions into the machines for processing. This process need to be effectively done with the consideration of cost minimization and meeting deadline to improve the user satisfaction. In the previous work, CRESP approach is used which focus on allocating the map reduces tasks in the machine with the consideration of reduction of cost and deadline. However this method does not concentrate on the skew and stragglers problem which can occur while handling the largest task. In our work, we try improve the performance of resource allocation strategy by considering the skews and stragglers problem in mind. This problem of skews and stragglers are handled by introducing the partitioning mechanism. The partitioning mechanism will improve the failure of task allocation strategy.**

Keywords—: Map reduce, Big data ,Cloud computing, Resource provisioning, Performance modeling

I.INTRODUCTION

BIG data is the term for a collection of data sets so large and complex that it becomes difficult to process using on-hand database management tools or traditional data processing applications. The challenges include capture, curation, storage, search, sharing, transfer, analysis, and visualization. The trend to larger data sets is due to the
additional information derivable from analysis of a single large set of related data, as compared to separate smaller sets with the same total amount of data, allowing correlations to be found to "spot business trends, determine quality of research, prevent diseases, link legal citations, combat crime, and determine real-time roadway traffic conditions.

Scientists regularly encounter limitations due to large data sets in many areas, including meteorology, genomics, connect omics, complex physics simulations, and biological and environmental research. The limitations also affect Internet search, finance and business informatics. Data sets grow in size in part because they are increasingly being gathered by ubiquitous information-sensing mobile devices, aerial sensory technologies (remote sensing), software logs, cameras, microphones, radio-frequency identification readers, and wireless sensor networks. Big data is difficult to work with using most relational database management systems and desktop statistics and visualization packages, requiring instead "massively parallel software running on tens, hundreds, or even thousands of servers". What is considered "big data" varies depending on the capabilities of the organization managing the set, and on the capabilities of the applications that are traditionally used to process and analyze the data set in its domain. "For some organizations, facing hundreds of gigabytes of data for the first time may trigger a need to reconsider data management options. For others, it may take tens or hundreds of terabytes before data size becomes a significant consideration.

B. OVERVIEW

Big data (also spelled Big Data) is a general term used to describe the voluminous amount of unstructured and semi-structured data a company creates -- data that would take too much time and cost too much money to load into a relational database for analysis. Although Big data doesn't refer to any specific quantity, the term is often used when speaking about petabytes and exa bytes of data. A primary goal for looking at big data is to discover repeatable business patterns. It's generally accepted that unstructured data, most of it located in text files, accounts for at least 80% of an organization's data. If left unmanaged, the sheer volume of unstructured data that's generated each year within an enterprise can be costly in terms of storage. Unmanaged data can also pose a liability if information cannot be located in the event of a compliance audit or lawsuit. Big data analytics is often associated with cloud computing because the analysis of large data sets in real-time requires a framework like Map Reduce to distribute the work among tens, hundreds or even thousands of computers.

C. HADOOP FILE SYSTEM

Apache Hadoop is an open-source software framework for storage and large scale processing of data-sets on clusters of commodity hardware. Hadoop is an Apache top-level project being built and used by a global community of contributors and users. It is licensed under the Apache License 2.0.

The Apache Hadoop framework is composed of the following modules :

- Hadoop Common - contains libraries and utilities needed by other Hadoop modules
- Hadoop Distributed File System (HDFS) - a distributed file-system that stores data on the commodity machines, providing very high aggregate bandwidth across the cluster.
- Hadoop YARN - a resource-management platform responsible for managing compute resources in clusters and using them for scheduling of users' applications.
- Hadoop Map Reduce - a programming model for large scale data processing.

All the modules in Hadoop are designed with a fundamental assumption that hardware failures (of individual machines, or racks of machines) are common and thus should be automatically handled in software by the framework. Apache Hadoop's Map Reduce and HDFS components originally derived respectively from Google's Map Reduce and Google File System (GFS) papers

Hadoop consists of the Hadoop Common package, which provides files system and OS level abstractions, a Map Reduce engine and the Hadoop Distributed File System (HDFS). The Hadoop Common package contains the necessary Java Archive (JAR) files and scripts needed to start Hadoop. The package also provides source code, documentation and a contribution section that includes projects from the Hadoop Community.

For effective scheduling of work, every Hadoop-compatible file system should provide location awareness: the name of the rack where a worker node is. Hadoop applications can use this information to run work on the node where the data is, and, failing that, on the same rack/switch, reducing backbone traffic. HDFS uses this method when replicating data to try to keep different copies of the data on different racks. The goal is to reduce the impact of a rack power outage or switch failure, so that even if these events occur, the data may still be readable. A small Hadoop cluster includes a single master and multiple worker nodes. The master node consists of a Job Tracker, Task Tracker, Name Node and Data Node. A slave or worker node acts as both a Data Node and Task Tracker, though it is possible to have data-only worker nodes and compute-only worker nodes. These are normally used only in nonstandard applications. Hadoop requires Java Runtime Environment (JRE) 1.6 or higher. The standard start-up and shutdown scripts require Secure Shell(ssh) to be set up between nodes in the cluster.

D. APPLICATIONS

The HDFS file system is not restricted to Map Reduce jobs. It can be used for other applications, many of which are under development at Apache. The list includes the HBase database, the Apache Mahout machine learning system, and the Apache Hive Data Warehouse system. Hadoop can in theory be used for any sort of work that is batch-oriented rather than real-time, that is very data-intensive, and able to work on pieces of the data in parallel. As of October 2009, commercial applications of Hadoop included:

- Log and/or clickstream analysis of various kinds
- Marketing analytics
- Machine learning and/or sophisticated data mining
- Image processing
- Processing of XML messages
- Web crawling and/or text processing

General archiving, including of relational/tabular data, e.g. for compliance

With the deployment of web applications, scientific computing, and sensor networks, a large amount of data can be collected from users, applications, and the environment. For example, user click through data has been an important data source for improving web search relevance and for understanding online user behaviors. Such datasets can be easily in terabyte scale; they are also continuously produced. Thus, an urgent task is to efficiently analyze these large datasets so that the important information in the data can be promptly captured and understood. As a flexible and scalable parallel programming and processing model, recently Map Reduce (and its open source implementation Hadoop) has been widely used for processing and analyzing such large scale datasets. On the other hand, data analysts in most companies, research institutes, and government agencies have no luxury to access large private Hadoop/Map Reduce clouds.

Therefore, running Hadoop/Map Reduce on top of a public cloud has become a realistic option for most users. In view of this requirement, Amazon has developed Elastic Map Reduce that runs on-demand Hadoop/Map Reduce clusters on top of Amazon EC2 nodes. There are also scripts1 for users to manually setup Hadoop/Map Reduce on EC2 nodes.

Computing devices have numerous uses and are essential for businesses, scientists, governments, engineers, and the everyday consumer. What all these devices have in common is the potential to generate data. Essentially, data can come from anywhere. Sensors gathering climate data, a person posting to a social media site, or a cell phone GPS signal are example sources of data. The popularity of the Internet alongside a sharp increase in the network bandwidth available to users has resulted in the generation of huge amounts of data. In response to these very same issues, engineers at Google developed the Google File System, a distributed file system architecture model for large-scale data processing and created the Map Reduce programming model.

Map Reduce is a programming model for processing large data sets with a parallel, distributed algorithm on a cluster. A Map Reduce program is composed of a Map() procedure that performs filtering and sorting and a Reduce() procedure that performs a summary operation. The Map Reduce System orchestrates by marshalling the distributed servers, running the various tasks in parallel, managing all communications and data transfers between the various parts of the system, providing for redundancy and fault tolerance, and overall management of the whole process. Map Reduce is a framework for processing parallelizable problems across huge datasets using a large number of computers (nodes), collectively referred to as a cluster or a grid.

Hadoop is an open source software implementation of Map Reduce, written in Java. Hadoop was created in response to the need for a Map Reduce framework that was unencumbered by proprietal licenses, as well as the growing need for the technology in Cloud computing. It focuses on Hadoop and investigates the load balancing mechanism in Hadoop's Map Reduce framework for small-to medium-sized clusters. This is an important area of research for several reasons. In addition to that to provide effective scheduling it use Map Reduce Task Scheduling algorithm for Deadline constraints. In this technique a node classification algorithm is used by measuring the node's computing capacity. Through this distribution of the data according to the node capacity. In addition to that it use micro-partioning methods to break the workload into many small tasks that are dynamically scheduled at runtime. This approach is only effective in systems with high -throughput, low-latency task schedulers and efficient data materialization.

II.LITERATURE SURVEY

A.CRESP: Towards Optimal Resource Provisioning for Map Reduce Computing in Public Clouds - Keke Chen, James Powers, Shumin Guo, and Fengguang Tian

Running Map Reduce programs in the cloud introduces this unique problem: how to optimize resource provisioning to minimize the monetary cost or job finish time for a specific job? The whole process of Map Reduce processing is studied and build up a cost function that explicitly models the relationship among the time cost, the amount of input data, the available system resources (Map and Reduce slots), and the complexity of the Reduce function for the target Map Reduce job. The problem of optimizing resource provisioning for Map Reduce programs involves two intertwined factors: the monetary cost of provisioning the virtual machine nodes and the time cost to finish the job.

However, resources are provisioned at cost, which are also related to the amount of time for using the resources. It propose a method to help users make the decision of resource provisioning for running Map Reduce programs in public clouds. This method, Cloud RESource Provisioning (CRESP) for Map Reduce Programs, is based on the proposed specialized Map Reduce time cost model that has a number of model parameters to be determined for a specific application.

B.Big table: A Distributed Storage System for Structured Data by Fay Chang, Jeffrey Dean

Big table is a distributed storage system for managing structured data that is designed to scale to a very large size: petabytes of data across thousands of commodity servers. Many projects at Google store data in Big table, including web indexing, Google Earth, and Google Finance. These applications place very different demands on Big table, both in terms of data size and latency requirements. Despite these varied demands, Big table has successfully provided a flexible, high-performance solution for all of these Google products. In this work it describe the simple data model provided by Big table, which gives clients dynamic control over data layout and format, and it describe the design and implementation of Big table..

Data Model

A Big table is a sparse, distributed, persistent multidimensional sorted map. The map is indexed by a row key, column key, and a timestamp; each value in the map is an uninterpreted array of bytes. It settled on this data model after examining a variety of potential uses of a Big table-like system. As one concrete example that drove some of this design decisions, suppose it want to keep a copy of a large collection of web pages and related information that could be used by many different projects; let us call this particular table the Web table.

C.Map Reduce: Simplified Data Processing on Large Clusters by Jeffrey Dean and Sanjay Ghemawa

Map Reduce is a programming model and an associated implementation for processing and generating large data sets. Users specify a map function that processes a key/value pair to generate a set of intermediate key/value pairs, and a reduce function that merges all intermediate values associated with the same intermediate key. Programs written in this functional style are automatically parallelized and executed on a large cluster of commodity machines. Map Reduce runs on a large cluster of commodity machines and is highly scalable: a typical Map Reduce computation processes many terabytes of data on thousands of machines. The Map Reduce library in the user program first splits the input files into M pieces of typically 16 megabytes to 64 megabytes (MB) per piece. It then starts up many copies of the program on a cluster of machines. One of the copies of the program is special . the master. The rest are workers that are assigned work by the master. There are M map tasks and R reduce tasks to assign. The master picks idle workers and assigns each one a map task or a reduce task A worker who is assigned a map task reads the contents of the corresponding input split.

III. PROBLEM DEFINITION

The tremendous growth in the computer networks enlarge the network bandwidth available to users has resulted in the generation of huge amounts of data. For a single computer the amount of data generated is too large so that the computation of the data takes more time and also increases the storage space. In order to reduce the computation time and decreases the storage space the workload is distributed on two or more computers. In addition to that distribution of workloads depends upon the node capacity so that it achieve the effective scheduling.

To distribute the workload among the nodes in the map reduce framework. In the map reduce frame work the workload is distributed and working on the data in parallel manner. The main objective of the project is to effectively distribute the workloads among the nodes and to reduce the storage space. In addition to that, it use node classification algorithm for enhance the scheduling.

It is very challenging to achieve proper distribution of workload among the nodes in the map reduce the framework. The main motivation of the project is due to the marvelous growth in the computer applications there is more workload in single computer causes large computation of data process and takes large memory space. In addition workload is distributed based on the node capacity so that the efficient scheduling is achieved.

IV. EXISTING SYSTEM

The existing work tries to find the best resource allocation for three typical situations:

1) With a certain limited amount of monetary budget;
2) With a time constraint;
3) and the optimal tradeoff curve without any constraint.

These objectives are reached in the existing by building the cost model priory. The cost model is used to calculate the each and every mapping of tasks. The costs that are occurred while executing the tasks are occurs in following stages. The first component is reading a block of data from the disk, which can be either local or remote data block. Let's assume the average cost is a function of the size of data block b: i(b). The second component is the user defined Map function, the complexity of which is determined by the input data size b, denoted as f(b). The Map function may output data in size of om(b) that might vary depending on the specific data. The output will be a list of <key, value> pairs. The result will be partitioned and sorted by the key into R shares for the R Reduce tasks. It denote the cost of partitioning and sorting with s(o_m(b), R). If the partitioning process uses a hash function to map the keys, the partitioning cost is independent of R.

A. *Disadvantages*

- The failures occurred due to the skew and stragglers concepts are taken into consideration which may violate the performance of the map reduce tasking programs.
- The storage will be consumed more in the existing work.

V. PROPOSED SYSTEM

Map Reduce is a programming model developed as a way for programs to manage with large amounts of data. It achieves this goal by distributing the workload among multiple computers and then working on the data in parallel.

Programs that execute on a Map Reduce framework need to divide the work into two phases:

- Map
- Reduce

Each phase has key-value pairs for both input and output . To implement these phases, a programmer needs to specify two functions:

- A map function called a Mapper
- Corresponding reduce function called a Reduce

When a Map Reduce program is executed on Hadoop(open source software), it is expected to be run on multiple computers or nodes. Therefore, a master node is required to run all the required services needed to coordinate the communication between Mappers and Reducers. An input file is then split up into fixed sized pieces called input splits. These splits are then passed to the Mappers who then work in parallel to process the data contained within each split. As the Mappers process the data, they partition the output. Each Reducer then gathers the data partition designated for them by each Mapper, merges them, processes them, and produces the output file.

It is the partitioning of the data that determines the workload for each reducer. In the Map Reduce framework, the workload must be balanced in order for resources to be used efficiently. An imbalanced workload means that some reducers have more work to do than others. This means that there can be reducers standing idle while other reducers are still processing the workload designated to them.

The micro-partitioning method is used in this work to divide the workload into smaller tasks. It introduce a new approach for avoiding skew and stragglers during the reduce phase. The key technique is to run a large number of reduce tasks, splitting the map output into many more partitions than reduce machines in order to produce smaller tasks. These tasks are assigned to reduce machines in a "just-in-time" fashion as workers become idle, allowing the task scheduler to dynamically mitigate skew and stragglers. Running many small tasks lessens the impact of stragglers, since work that would have been scheduled on slow nodes when using coarse-grained tasks can now be performed by other idle workers. With large tasks, it can be more efficient to exclude slow nodes rather than assigning them any work. By assigning smaller units of work, jobs can derive benefit from slower nodes.

A. *Advantages*

- Storage space can be reduced effectively
- More Accuracy
- Reduce the delay
- Micro-partioning is used
- This approach is only effective in systems with high-throughput, low-latency

VI. EXPERIMENTS

A.Node classification method

In the node classification algorithm, it use the comprehensive criterion to represent the data processing speed. This criterion should be the comprehensive processing capacity facing all memory intensity, IO intensity and CPU intensity jobs. In a heterogeneous environment, cluster usually contains nodes with different processing capacity, which means the speed of the node

processes data. It classify the nodes according to this processing capacity. There are two purposes of node classification with their processing capacity: one is to optimize the data distribution in order to improve the data locality; the other is to improve the evaluation accuracy of the task remaining time in heterogeneous environment.
In order to get the differences between nodes with their capacities, it divide the nodes to different levels. In the cluster, it take the slowest data processing speed nodes as the first level and the level factor is 1The represent the *p*-th level of servers' processing capacity, which is recursive calculated by multiplying and the classification factor . So it known that is larger than . And the nodes that belong to has a better capacity than the nodes that belong to . According to the definition of it can separate the heterogeneous environment nodes to different levels simply. To quantize the node's computing capacity simply by running a group of specific tasks. On a given cluster, it set each node with one Map slot and one Reduce slot. And all the map tasks are feed by the same input data. Then the benchmark jobs run on the cluster It record the completion time of each task on where it ran. It set as a constant value larger than 1. The node classification algorithm enables us to decide the nodes' level by their computing capacity. The data distribution strategy is that the size of each node's data is in proportion to the node's level.

B. *Learn the Cost Models*

Map Reduce processing is a mix of sequential and parallel processing. The Map phase is executed before the Reduce phase. However, in each phase many Map or Reduce tasks are executed in parallel. To clearly describe the MapReduce execution, it would like to distinguish the concepts of Map/Reduce slot and Map/Reduce task. Each Map (or Reduce) task is executed in a Map (or Reduce) slot. A slot is a unit of computing resources allocated for running the tasks. According to the system capacity, a computing node can only accommodate a fixed number of slots so that the tasks can be run in the slots in parallel without serious competition. In Hadoop, the Tasktracker running in each slave node has to set the number of Map slots and the number of Reduce slots. A common setting for a multi-core computer is to have one Map or Reduce slot per core. Without loss of generality, let's assume there are m Map slots and r Reduce slots in total over all slave nodes. With the formulation of the cost function in terms of input variables M, m, and R, it need to learn the parameters βi. Note that βi should be different from application to application because of data distributions, specific I/O patterns, and data processing logic. It design a learning procedure as follows. First, for a specific Map Reduce program, it randomly sample the variables M, m, and R from certain ranges. For example, m and R (i.e., r) are chosen within 50; M is chosen so that at least two rounds of Map tasks are available for testing. Second, it collect the time cost of the test run of the Map Reduce job for each setting of (M, m,R), which forms the training dataset. Third, regression modeling is applied to learn the model from the training data with the transformed variables

$x_1 = M = m$, $x_2 = MR/m$, $x_3 = m/R$;
$x_4 = (M \log M) / R$, $x_5 = M/R$, $x_6 = M$, $x_7 = R$

Because βi are the weights of the components in the total cost, it have the constraints βi >= 0 . . . r, which require nonnegative linear regression to solve the learning problem. The cross-validation method is then used to validate the performance of the learned model.

C. *Optimized resource Allocation*

With the cost model it are now ready to find the optimal settings for different decision problems. It try to find the best resource allocation for three typical situations: 1) with a certain limited amount of monetary budget; 2) with a time constraint; 3) and the optimal tradeoff curve without any constraint. In the following, it formulate these problems as optimization problems based on the cost model. In all the scenarios it consider, it assume the model parameters βi have been learned with sample runs in small scale settings. For the simplicity of presentation, it assume the simplified model T2 is applied. Since the input data is fixed for a specific Map Reduce job, M is a constant. It also consider all general Map Reduce system configurations have been optimized via other methods and fixed for both small and large scale settings.

In Map Reduce, the job's execution progress includes Map and Reduce stage. So, the job's completion time contains Map execution time and Reduce execution time. In view of the differences between Map and Reduce's code, it divide the scheduling progress into two stages, namely Map stage and Reduce stage. In the aspect of the task's scheduling time prediction, the execution time of Map and Reduce is not correlative; their execution time depends on the input data and function of their own. Therefore, in this work the scheduling algorithm sets two deadlines: map-deadline and reduce-deadline. And reduce-deadline is just the users' job deadline. In order to get map-deadline, it need to know the Map task's time proportion on the task's execution time. In a cluster with limited resources, Map slot and Reduce slot number is decided. For an arbitrary submitted job with deadline constraints, the scheduler has to schedule reasonable with the remaining resources in order to assure that all jobs can be finished before the deadline constraints. According to map-deadline, it can acquire the current map task's slot number it needs; and with reduce-deadline, it can get the current reduce task's slot number it needs. task's slot number it needs; and with reduce-deadline, it can get the current reduce task's slot number it needs.
In the task scheduling , to get map-deadline, it need to know the Map task's time proportion on the task's execution time. In a cluster with limited resources, Map slot and Reduce slot number is decided. For an arbitrary submitted job with deadline constraints, the scheduler has to schedule reasonable with the remaining resources in order to assure that all jobs can be finished before the deadline constraints. The scheduling strategy of MSTD is based on and . The minimum Map and Reduce slot number required of job *J* can be denoted as and respectively. The symbol reflects that Map tasks should be scheduled at present in order to meet job *J* 's map-

deadline, as well as to meet the reduce-deadline Reduce tasks should be scheduled. In the scheduling process, it take and as the basic criteria of priority allocation. At the beginning of the job be submitted, there is no data available, so the scheduler can't estimate the required slots or the completion time of tasks. In this case, the job's precedence is over than the others. In some scenarios, jobs may have already missed their deadline. The strategy it use is the same as the previous case: set such jobs' tasks with the highest priority.

C. *Micro-Partioning Method*

The key technique is to run a large number of reduce tasks, splitting the map output into many more partitions than reduce machines in order to produce smaller tasks. These tasks are assigned to reduce machines in a "just-in-time" fashion as workers become idle, allowing the task scheduler to dynamically mitigate skew and stragglers. Running many small tasks lessens the impact of stragglers, since work that would have been scheduled on slow nodes when using coarser-grained tasks can now be performed by other idle workers. With large tasks, it can be more efficient to exclude slow nodes rather than assigning them any work. By assigning smaller units of work, jobs can derive benefit from slower nodes. Micro-tasks can also help to mitigate skew.

For inputs containing few distinct keys, fine-grained partitioning may result in many empty reduce tasks that receive no data. These empty reduce tasks are unproblematic, since they can be easily detected and ignored by the scheduler. Jobs with few distinct keys are the most sensitive to partitioning skew, since there may not be enough other work to mask the effects of a straggling task created by a key collision in the hash partitioning function. For jobs with large numbers of distinct keys, the impact of key collisions is small. Micro-tasks do not specifically address record size or computational skew, but these types of skew must be handled in an application-specific manner. Micro-tasks help to achieve a more even assignment of indivisible units of work; techniques for reducing the units of work, such as map-side combining, are complimentary and orthogonal to this approach.

E. Performance Evaluation

Finally in this module the performance of the existing and the proposed approaches were illustrated and evaluated. In the existing system based on cost analysis function, the optimized resource allocation in done. In the proposed system, the micro-partioning method is also considered before allocating the tasks into the resources. Compared to existing method there is it achieve efficient scheduling in the proposed system and achieve high throughput.

VII. CONCLUSION

In this work, it study the components in Map Reduce processing and build a cost function that explicitly models the relationship among the amount of data, the available system resources (Map and Reduce slots), and the complexity of the Reduce function for the target Map Reduce program. The model parameters can be learned from test runs. Based on this cost model, it can solve a number of decision problems, such as the optimal amount of resources that can minimize the monetary cost with the constraint on monetary budget or job finish time. To improve the load balancing for distributed applications, micro partitioning techniques. By improving load balancing, Map Reduce programs can become more efficient at handling tasks by reducing the overall computation time spent processing data on each node. In addition to that it use Map Reduce Task Scheduling algorithm for cost and time constraints for the efficient scheduling. For that it use node classification method and distribute the workload among the nodes according to the node capacity. After that a micro partitioning method is used for applications using different input samples. This approach is only effective in systems with high-throughput, low-latency task schedulers and efficient data materialization.

In the future, it would like to implement the proposed task scheduler architecture and perform additional experiments to measure performance using straggling or heterogeneous nodes. It also plan to investigate other benefits of micro-tasks, including the use of micro-tasks as an alternative to preemption when scheduling mixtures of batch and latency-sensitive jobs.

REFERENCES

[1]. Keke Chen, James Powers, Shumin Guo, and Fengguang Tian:CRESP: Towards Optimal Resource Provisioning for MapReduce Computing in Public Clouds, pp 1403-1412

[2]. Chang F, Dean J, Ghemawat S, Hsieh WC, Wallach DA, Burrws M, Chandra T, Fikes A, Gruber RE (2006) Big table: a distributed storage system for structured data. In: 7th UENIX symposium on operating systems design and implementation, pp 205–218.

[3]. Dean J, Ghemawat Dean S (2008) MapReduce: simplified data processing on large clusters. Commun ACM 51:107–113.

[4]. Ghemawat S, Gobioff H, Leung S-T (2003) The Google file system. In: 19th ACM symposium on operating systems principles (SOSP).

[5]. Jiang W, Agrawal G (2011) Ex-MATE data intensive computing with large reduction objects and its application to graph mining. In: IEEE/ACM international symposium on cluster, cloud and grid computing, pp 475–484.

[6]. Jin C, Vecchiola C, Buyya R (2008) MRPGA: an extension of MapReduce for parallelizing genetic algorithms. In: IEEE fourth international conference on escience, pp 214–220.

[7]. Kavulya S, Tany J, Gandhi R, Narasimhan P (2010) An analysis of traces from a production MapReduce cluster. In: IEEE/ACM international conference on cluster, cloud and grid computing, pp 94–95.

[8]. Krishnan A (2005) GridBLAST: a globus-based high-throughput implementation of BLAST in a grid computing framework. Concurr Comput 17(13):1607–1623.

[9]. Liu H, Orban D (2011) Cloud MapReduce: a MapReduce implementation on top of a cloud operating system. In: IEEE/ACM international symposium on cluster, cloud and grid computing, pp 464–474.

[10]. Hsu C-H, Chen S-C (2012) Efficient selection strategies towards processor reordering techniques for improving data locality in heterogeneous clusters. J Supercomput 60(3):284–300
Pages 89 – 98, 2006