Use of MapReduce for Data Mining and Data Optimization on a Web Portal

Christopher A. Moturi
School of Computing and Informatics
University of Nairobi
Kenya

Silas K. Maiyo
School of Computing and Informatics
University of Nairobi
Kenya

ABSTRACT
This paper studied the design, implementation and evaluation of a MapReduce tool targeting distributed systems, and multi-core system architectures. MapReduce is a distributed programming model originally proposed by Google for the ease of development of web search applications on a large number of clusters of computers. We addressed the issues of limited resource for data optimization for efficiency, reliability, scalability and security of data in distributed, cluster systems with huge datasets. The study's experimental results predicted that the MapReduce tool developed improved data optimization. The system exhibits undesired speedup with smaller datasets, but reasonable speedup is achieved with a larger enough datasets that complements the number of computing nodes reducing the execution time by 30% as compared to normal data mining and processing. The MapReduce tool is able to handle data growth trendily, especially with larger number of computing nodes. Scaleup gracefully grows as data and number of computing nodes increases. Security of data is guaranteed at all computing nodes since data is replicated at various nodes on the cluster system hence reliable. Our implementation of the MapReduce runs on distributed cluster computing environment of a national education web portal and is highly scalable.

General Terms
Data Mining, Data Optimization, Distributed Cluster Computing, Multiprocessor Systems

Keywords
MapReduce, Hadoop, Scalability

1. INTRODUCTION
Data and information explosion propelled by the exponential growth in digitized data is an unstoppable reality. To be able to extract relevant and useful knowledge from this voluminous data in order to make well-informed decision is a competitive advantage in the current information age. In almost all sectors, like education, datasets have grown from gigabyte to terabyte, and now headed to petabyte. As a result, parallel and distributed computing is often strongly sought after to alleviate these challenges. The need for data optimization, measuring performance efficiency, reliability, scalability and effectiveness of parallel models, cluster and grid computing is apparent.

It is envisioned that there is very huge data warehousing in different education databases at various locations and networks, in distributed, cluster, computing systems. Data access, retrieval, and analysis is expected to be a big challenge for educationists. Therefore a MapReduce tool is required to easily and effectively optimize the process of delivering the required information to the consumers in a reliable and scalable environment with concern on the security.

The objectives of this research were to explore the usability of the MapReduce programming technique on a multi-core shared memory system and cluster computing; Optimization of MapReduce tool developed to improve its performance and scalability; Implications of using Java threading library to attain parallelism on multi-core systems; and Effective handling of failures on parallel execution of tasks.

The case of a national education portal in Kenya, a teachers' resource center to support research and development, was used to test the tool developed.

2. LITERATURE REVIEW
2.1 MapReduce in Distributed Systems
MapReduce is a programming model and an associated implementation for processing and generating large datasets that is amenable to a broad variety of real-world tasks (Dean J. et al, 2004). The MapReduce paradigm of parallel programming provides simplicity, while at the same time offering load balancing and fault tolerance The Google File System (GFS) that typically underlies a MapReduce system provides the efficient and reliable distributed data storage needed for applications involving large databases (Ghemawat S., et al, 2003). The marriage of these systems, is typical in their deployment, and represents the necessary confluence of data distribution and parallel computation.

MapReduce is inspired by the map and reduces primitives present in functional languages. In its pure form, various implementations of the MapReduce interface are possible, depending on the desired context. Some currently available implementations are: shared-memory multi-core system (Ranger, et al., 2007) and (Yoo, et al., 2009), asymmetric multi-core processors (Rafique, et al. 2009), graphic processors (He, et al., 2008), and cluster of networked machines (Dean J., et al., 2008). The most popular implementation is probably the one introduced by Google, which utilizes large clusters of commodity computers connected with switched Ethernet. In essence, the Google's MapReduce technique simplifies the development and lowers the cost of large-scale distributed applications on clusters of commodity machines.

Although the distributed computing is largely simplified with the notions of Map and Reduce primitives, the underlying infrastructure is non-trivial in order to achieve the desired performance. A key infrastructure in Google's MapReduce is the underlying distributed file system to ensure data locality and availability. Google's MapReduce implementation leverages and depends heavily on an in-house distributed file system known as Google File System (Ghemawat S., et al, 2003). Combining the MapReduce programming technique and an efficient distributed file system, one can easily achieve the goal of distributed computing with data parallelism over thousands of computing nodes; processing data on terabyte

and petabyte scales with improved system performance, optimization and reliability.

2.2 Hadoop Distributed File System

This research adapted a MapReduce tool in the distributed system making use of a popular open source implementation called Hadoop. The Hadoop Distributed File system (HDFS) was designed to provide high throughput access with reliable storage of very large files across machines in a large cluster (Apache Hadoop, Online, 2011). It is designed for high performance and thus not as general-purpose as the commonly used distributed file systems such as Network File System (NFS) and Common Internet File System (CIFS). Applications that use HDFS are assumed to perform long sequential streaming reads from files and databases. HDFS is optimized to provide streaming read performance, sacrificing the random seek times to arbitrary positions in files. Due to the large size of files, and the sequential nature of reads, the system does not provide mechanism for local data caching because the overhead of caching warrants a re-read from the HDFS.

3. RESEARCH METHODOLOGY

Using the Agile Methodology a MapReduce tool was designed and developed on the Linux platform. Each component in the tool was developed iteratively; each iteration involved designing, coding, testing and integration.

The Hadoop MapReduce tool uses a master/slave architecture that is similar to the one employed in HDFS. The master, called JobTracker, is responsible for:

(a) Querying the NameNode for the locations of the data block involved in the Job,

(b) Scheduling the computation tasks (with consideration of the block locations retrieved from the NameNode) on the slaves, called TaskTrackers, and

(c) Monitoring the success and failures of the tasks.

Hadoop was designed to have high degree of fault tolerance. In comparison to many available parallel/distributed systems, it is able to complete the assigned tasks failures in the cluster (Pavlo, A. et al, 2009). The primary way that Hadoop achieves fault tolerance is through restarting tasks. The slave nodes involved in the computation are in constant communication with the master node (JobTracker). If a TaskTracker failed to communicate with the JobTracker for a period of time (by default, 1 minute), the JobTracker assumed failure on that TaskTracker. It then assigned another active TaskTracker to re-execute all the tasks that were in progress on the failed Tasktracker.

The input for a Hadoop MapReduce task was typically very large files residing in the HDFS. The format of these input files is arbitrary; it could be formatted text file, binary format or any user-defined format.

MapReduce tool is capable of processing huge datasets on certain kind of distributable problems using a large number of computers (nodes), collectively referred to as a cluster or as a grid. Computational processing can occur on data stored either in a file system (unstructured) or within a database (structured).

"Map" step: The master node takes the input, partitions it up into smaller sub-problems, and distributes those to worker nodes. A worker node may do this leading to a multi-level tree structure. The worker node processes that smaller problem, and passes the answer back to its master node.

"Reduce" step: The master node then takes the answers to all the sub-problems and combines them in some way to get the output — the answer to the problem it was originally trying to solve.

In a logical view, the Map and Reduce functions of MapReduce are both defined with respect to data structured in (key, value) pairs. Map takes one pair of data with a type in one data domain, and returns a list of pairs in a different domain:

$$\text{Map } (k1, v1) \rightarrow \text{list } (k2, v2)$$

The Map function is applied in parallel to every item in the input dataset. This produces a list of (k2,v2) pairs for each call. After that, the MapReduce tool collects all pairs with the same key from all lists and groups them together, thus creating one group for each one of the different generated keys.

The Reduce function is then applied in parallel to each group, which in turn produces a collection of values in the same domain:

$$\text{Reduce } (k2, \text{list } (v2)) \rightarrow \text{list } (v3)$$

Each Reduce call typically produces either one value v3 or an empty return, though one call is allowed to return more than one value. The returns of all calls are collected as the desired result list.

Once the MapReduce tool was developed as explained above, it was placed at the server where the central data bank of education web portal is situated and also at the client computers (computing nodes) where the portal could be accessed by users. The data mining and data optimization executions were observed on the existing massive data in the education web portal case where many computing nodes are accessed by several users simultaneously, where observations were made and analysis gave a true picture of data mining and data optimization as explained in the under results section.

4. RESULTS AND DISCUSSION

4.1 Results

The study realized the development, experimenting and testing of a MapReduce tool in a distributed cluster computing environment. The results realized at the education web portal case of study were encouraging in view of the objectives of the study. However, there were certain aspects culminating from the research that were not in tandem with the inclination of the project, that is, being undertaken as a result of data optimization. The instruments used which involved a procedure were able to capture details that included observations below.

It was observed that the MapReduce tool is much efficient in data optimization and very reliable since it reduces the time of data access or loading by more than 50%.

The MapReduce tool could provide availability of jobs, tasks, and log history file which then improves the performance of the MapReduce tool and data access and processing in general.

Data is distributed to the computing nodes at equally faster speed across all the nodes making the process more secured and efficient at all levels and nodes, irrespective of external challenges.

Attaining scalability was a major breakthrough since growth is inevitable at all times. It was found that with MapReduce tool you can add extra nodes which are allocated Job IDs and linked to Job Tasks at any place and at any time of the processing which can be effected on real time basis. This therefore provides an efficient way of scaling up the number of nodes on the cluster system without losing connections.

Experiments were conducted to evaluate the computation performance of the MapReduce prototype tool. The performance evaluation was based on the measurement of **speedup** (to address the efficiency and reliability of the tool), **scaleup** (to address the scalability of the tool), and the **sizeup** (to address the ability of parallelism to handle data growth).

The experiments were also compared with other researchers' evaluations done earlier on cluster machines such as "Amazon Elastic Compute Cloud (EC2)", which operates a "pay as you go" model, allowing flexibility of paying only the required computation time.

The experiments were conducted with a cluster of 1, 2, 4, 8, and 16 computing nodes configured as a cluster computing with hadoop distributed file system. The initial input dataset used contained huge files of different formats and types from Kenya education web portal.

The experiments were obtained from the average of 5 runs. Each run executed the entire data mining process, Mapping, splitting/shuffle/combine, and Reducing attempts and producing the output which was a selected field's content with a computed answer.

A summary of the experiment results are presented in Table 1. The table shows the average execution time (in seconds) of one run under each experiment scenario.

Table 1: Experimental Results

Execution Time (sec) / Data Size (count)	Number of Nodes				
	1	2	4	8	16
1x (148)	7	5	4	3	1
2x (296)	12	8	5.4	5	4.2
4x (592)	15	9	7	6	5
8x (1184)	20	11	8	7	7
16x (2368)	28	15	9	8	8
32x (4736)	42	20	12	10	9

4.2 Discussions

Three measurements were used for the evaluations: speedup, scaleup, and sizeup. The following sub-sections discuss these measurements and the results obtained.

4.2.1 Speedup

Speedup tries to evaluate the ability of the parallelism to improve the execution time. It is defined as the ratio of the sequential execution time to the parallel execution time. It is expressed as follows:

Speedup (m) = T(1) / T(m)

Where m is the number of computing node, T(1) is the execution time of the tasks on 1 computing node, and T(m) is the execution time of the parallel task with m computing node. A perfect parallelism demonstrates linear speedup, i.e. a system with m times the number of computers/nodes yields a speedup of m. However, linear speedup is difficult to achieve because the communication cost increases with the number of computing nodes.

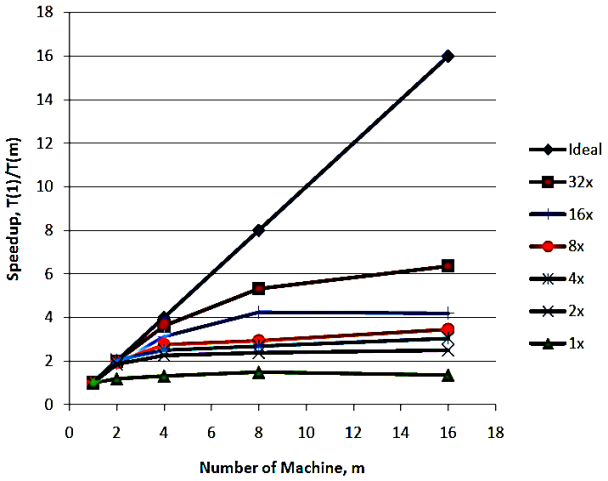

Figure 1: Speedup of the MapReduce Prototype System

From the results obtained, the system exhibits undesired speedup with smaller datasets, but reasonable speedup is achieved with a larger enough datasets that complements the number of computing nodes.

At 32x of initial input size, the speedup achieved with 8 machines is 5.32; but with 16 machines, the speedup achieved is only 6.36, a rather disappointing result.

Based on the trend shown in Figure 1, we could make two observations:

1 MapReduce tool is most suited for distributed computing with huge datasets;

2 Some rate-limiting factors exist in the system.

4.2.2 *Scaleup*

Scaleup evaluates the ability of parallelism to grow both the MapReduce system and the data size, that is, the scalability of the MapReduce tool. Scaleup is defined as the ability of an m-times larger system to perform an m-times larger job in the same run-time as the original system. Scaleup can be expressed as follows:

Scaleup (m) = T(1,D) / T(m, mD)

Where m is the number of computing node, T(1,D) is the execution time of the tasks on 1 computing node with data size of D, T(m, mD) is the execution time of the parallel tasks with m computing nodes with data size m times of D. A perfect parallelism demonstrates a constant scaleup with increasing number of computing nodes and data size.

Figure 2 shows the Scaleup of the prototype with the number of computing nodes and input data size increasing at a factor of 2, i.e. 1, 2, 4, 8 and 16.

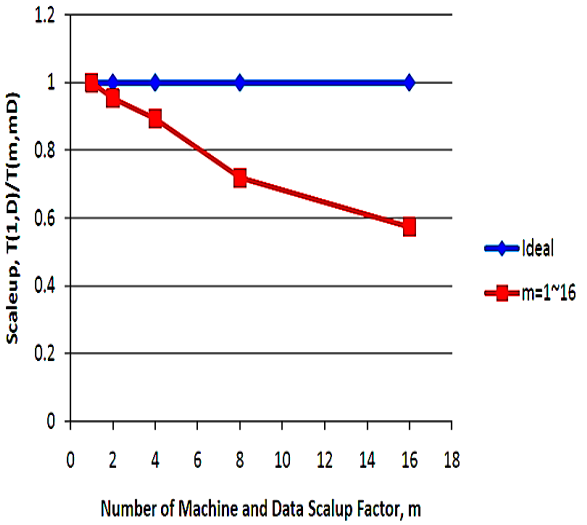

Figure 2: Scaleup of MapReduce Prototype System

From the results, we can conclude that the MapReduce tool exhibits good scaleup at 0.95, 0.89 and 0.72 with 2, 4 and 8 computing nodes respectively. With a 16 computing nodes cluster, the scaleup is slightly lower, measured at 0.57. The reason for such scaleup degradation is due to the increase task setup time and communication cost with growing number of computing nodes, which is expected scenario in distributed computing. Nevertheless, this shows that MapReduce is still considerably scalable with growing data and growing number of computing nodes, a key consideration for data intensive applications.

4.2.3 Sizeup

Sizeup evaluates the ability of the parallelism to handle growth. It measures how much longer it takes to execute the parallel tasks, when the data size is n-times larger than the original datasets. Sizeup analysis holds the number of computing node constant and grows the size of the datasets by the factor n. Sizeup can be expressed as follows:

Sizeup (m, n) = T(m, nD) / T(m, D)

Where m is the number of computing node and n is the incremental factor of the data size. T(m, D) is the execution time of the parallel tasks with m computing node and data size D and T(m, D) is the execution time of parallel tasks with m computing node and with data size n times of D.

Figure 3 shows the sizeup of the prototype. Each line corresponds to the sizeup behavior of a cluster of m computing nodes with increasing data load at a factor of n.

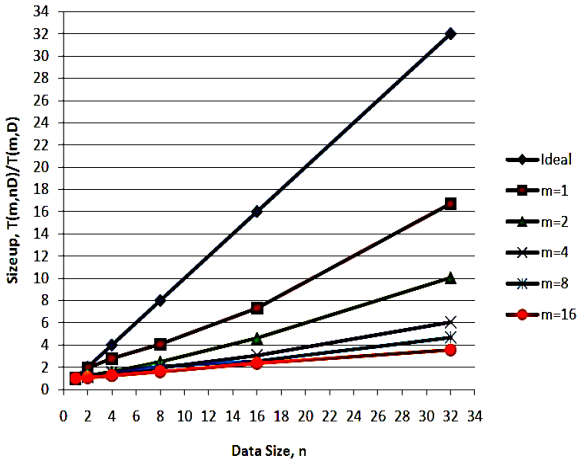

Figure 3: Sizeup of MapReduce Prototype System

The results showed that the MapReduce system is able to handle data growth at all configurations. The sizeup decreases with increasing number of computing nodes. With a 16 nodes cluster at 32 times the initial data, the execution time increases only by a factor of 3.55. This shows that MapReduce is able to handle data growth gracefully, especially with larger number of computing nodes.

4.2.4 Comparative Results

Our results compares with the various results output during the experimental study, including the evaluations of the Speedup, Sizeup and Scaleup ability of the parallelism to handle growth. Table 2 shows the various levels of performance computation of measurements at different quantity of cluster computing nodes.

Table 2: Speedup, Scaleup and Sizeup comparative results

Cluster Nodes	Scaleup	Sizeup	Speedup
2	0.95	2	2.06
4	0.89	3	3.86
8	0.72	4	5.32
16	0.59	7.86	6.36

Figure 4, shows that the scaleup of a cluster computing using the MapReduce technique reduces as the sizeup drastically increases with number of nodes.

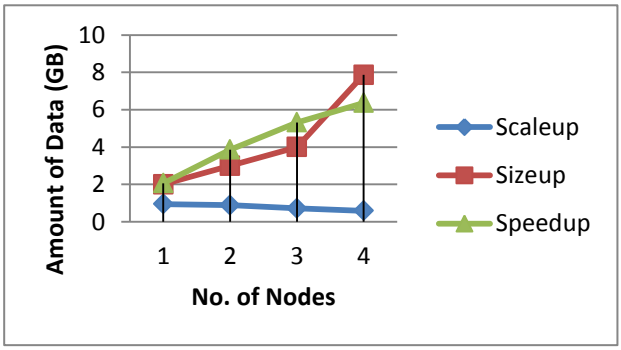

Figure 4: Comparative results for Speedup, Scaleup and Sizeup. Source: Researcher

The speedup increases significantly as the sizeup of the cluster grows inversely to the scaleup. It is evidently shown that this is unreliable and inefficient, expensive when the same computation is done in a cluster without the use of Hadoop MapReduce technique in a distributed file system as depicted by the Amazon Elastic Compute Cloud (EC2) system as studied in literature review.

5. CONCLUSIONS

The study successfully developed and implemented a MapReduce tool for data mining and data optimization using cheaply and easily available tools and procedures. The MapReduce tool was developed particularly for multi-core shared memory systems, taking advantage of current trends in parallelism. The efficiency of the tool highly depended on the granularity of the tasks executed by the worker threads and the usability of the API by maintaining a simple and restricted interface. Even if the framework were able to attain a speedup of 30%, it would still be beneficial to the researcher as it provides simple and efficient auto-parallelism. It is evident that the advantages provided by the tool under study, would clearly outweigh the disadvantages associated with it.

Achievement of scalability in the clustering environment was a major milestone. This was attested to be easily achieved and sustainable since additional nodes could easily be added on the cluster without negatively affecting the performance of other nodes on the cluster hence the cluster could easily grow and increase resource utilization.

The security of data is confident than usual operations since data is replicated across on all the nodes equally and in the event of destruction of one node, data can be recovered from other working nodes.

The evaluation results show that the prototype exhibits satisfactory performance in the parameters such as speedup, good scaleup, and reasonable sizeup. A key observation made in the evaluation was that the input data size is an important factor in achieving reasonable speedup. This is due to the fact that for optimal parallelism, one has to ensure the effects of overheads (such as inter-machines communications) in the parallelism is minimized by the computation time; and in most cases, large input size maximizes the computation time. The unexpected speedup performance leads to a detailed analysis of the results and it suggests that, although inter-machines communications and input size are factors in limiting ideal parallelism, the non-parallelizable portion in the individual computation task also plays a paramount role in limiting speedup. The evaluation also shows that Hadoop MapReduce exhibits good scalability with growth in both data size and number of computing nodes.

There is still need for continuous research in this field to develop more enhanced tools, systems and frameworks of MapReduce techniques implemented on Hadoop distributed file system in distributed systems.

Further work on improving this methodology should be considered by making use of latest tools, procedures and technologies such as in the Cloud Computing to solve issues of data optimization, reliability, scalability and data security, among other issues fault tolerance in cluster computing.

The study should be expanded to research on the impact of performance overheads and how to conquer them for better performance.

6. ACKNOWLEDGMENTS

Sincere acknowledgement to Lawrence Muchemi, Peter Wagacha, Dan Orwa and Joseph Ogutu, all from the School of Computing and Informatics, University of Nairobi, Kenya, who have contributed towards this paper.

Also to VVOB Kenya, National ICT Innovations and Integration Centre (NI3C) for allowing us to use their data among other resources.

7. REFERENCES

[1] Apache Hadoop. [Online] [Cited: 07 05, 2011.] http://hadoop.apache.org/

[2] Dean J. and Ghemawat S. 2004. “Mapreduce: Simplified Data Processing On Large Clusters,” In Proceedings of OSDI’04: 6th Symposium on Operating System Design and Implementation.

[3] Dean J. and Ghemawat S. 2008. “MapReduce: Simplified Data Processing on Large Clusters”. Communications of the ACM. Vol. 51, 1, pp. 107-113.

[4] Ghemawat S., Gobioff H., and Leung S.T. 2003. “The Google File System”. Proceedings of 19th ACM Symposium on Operating Systems Principles, pp 29-43

[5] Google and IBM Announce University Initiative to Address Internet-Scale Computing Challenges. Google Press Center. [Online] 10 08, 2007. [Cited: 07 05, 2011.]

[6] He, B., Fang, W., Luo, Q., Govindaraju, N. K., Wang, T. 2008. “Mars: A MapReduce Framework on Graphics Processors”. Proceedings of the 17th International Conference on Parallel Architectures and Compilation Techniques, pp. 260-269.

[7] Pavlo, A et al. 2009. “A Comparison of Approaches to Large-Scale Data Analysis”. Proceedings of the 35th SIGMOD International Conference on Management of Data, pp. 165-178

[8] Rafique, Mustafa. M. 2009. “Supporting MapReduce on Large-Scale Asymmetric Multi-Core Clusters”. ACM SIGOPS Operating Systems Review, Vol. 43, 2, pp. 25-34.

[9] Ranger, C., Raghuraman, R., Penmetsa, A., Bradski, G., Kozyrakis, C. 2007. “Evaluating MapReduce for Multi-core and Multiprocessor Systems”. Proceedings of the 2007 IEEE 13th International Symposium on High Performance Computer Architecture, pp. 13-24.

[10] Yoo, R. M., Romano, A.K. and Kozyrakis, C. 2009. Phoenix Rebirth: “Scalable MapReduce on a Large-Scale Shared-Memory System”. Proceedings of the 2009 IEEE International Symposium on Workload Characterization, pp. 198-207.