In-Map/In-Reduce: Concurrent Job Execution in MapReduce

Muhammad Idris
Ubiquitous Computing Laboratory
College of Electronics and Information
Kyung Hee University
Republic of Korea
Email: idris@oslab.khu.ac.kr

Shujaat Hussain
Ubiquitous Computing Laboratory
College of Electronics and Information
Kyung Hee University
Republic of Korea
Email: shujaat.hussain@oslab.khu.ac.kr

Sungyoung Lee
Ubiquitous Computing Laboratory
College of Electronics and Information
Kyung Hee University
Republic of Korea
Email: sylee@oslab.khu.ac.kr

***Abstract*—Hadoop based MapReduce(MR) has emerged as big data processing mechanism in terms of its data intensive applications. In data intensive systems; analysis and visualizations as a result of various algorithms can lead to differentiable and comparable results. Current implementations of MR facilitates to reuse the results of MR jobs in other MR jobs and to distribute the cloud resources among jobs. However, very little work is done in terms of using same data for multiple algorithms at the same time in a single job using either shared resources or dynamic resource allocation based on the data and scheduling of MapReduce jobs. In this paper we propose a method to execute multiple algorithms on same data in HDFS concurrently and to use the same available resources by dynamically managing the task assignment and results aggregation. Our proposed approach reduces the execution time and supports multiple algorithms execution in parallel. In-Map/In-Reduce shows 200% decrease in execution time.**

***Keywords*—Big Data, Hadoop, MapReduce, Data Intensive Computing, HDFS.**

I. Introduction

In the recent times data is increasing exponentially, requiring processing and analysis, termed as Big Data [1]. It demands storage (provided by Hadoop [2]), processing mechanism in less time and computational cost compared to the traditional applications as MapReduce [3]. Hadoop Distributed File System (HDFS) is developed to handle the data on clusters of commodity hardware as storage mechanism and MapReduce is a processing mechanism to process HDFS data. MapReduce algorithm provides the facility to write applications in a high level programming model and to hide the details of working of a program on a cloud (has many clusters) of commodity hardware. It works in two phases i.e., map and reduce. MapReduce runs a single algorithm on distributed data in parallel[4] as a single job for data intensive[5] applications. Data intensive applications tend to process large data in a distributed, decoupled and low inter-node dependent environment where data is the focus. In compute intensive applications, extensive computation on shared data with high coupling and message passing between individual workers takes place e.g., image processing, weather information processing using High Performance Computing(HPC). In an effort to combine data intensive solution along with compute intensive solution [6], we propose In-Map/In-Reduce concurrent job execution. Our motivation is to execute multiple MapReduce algorithms on same distributed data in a single MapReduce job e.g., multiple machine learning algorithms on same weather data. A motivational example would be predictive analysis where the same set of Big Data will be used to train multiple statistical models such as LM, GLM, ARIMA and different computation will be performed on same set of data or data split. We provide and implement In-Map and In-Reduce parallelization in MapReduce along with parallel data processing.

Apache open-source implementation of Google's MapReduce called Hadoop, facilitates running a single algorithm as single job on the cluster at a time i.e. parallel data processing. Efforts made in the big data over cloud computing and parallel computing community focuses on Hadoop to make it able to support parallel computation as well. The research is mainly focused on data communication between nodes in the cloud[7], data locality [8], job scheduling [9] and skew mitigation [10]. In job scheduling; an attempt is made to execute reduce tasks in parallel to the map tasks when some of map tasks have nished and some are still running. This is designed only for single algorithm running as MapReduce job.

MapReduce is basically a parallel data processing framework as it is mostly adopted by the Big Data community and the research in the community is mostly focused on optimization of MapReduce in its parallel data processing mechanism[4], MapReduce performance in homogeneous and heterogeneous [11] environments and scheduling in MapReduce. It will be a step forward to extend the MapReduce towards computation intensive systems.

In this paper, we propose methodology of parallel computation with data to make MapReduce both data and computation intensive framework. It gives the concept of In-Map and In-Reduce parallel processing in a single job. Hadoop is well suited for running single algorithm as single MapReduce job on the available data in the cluster; while our method exploits the ability of the cloud and attempts to use available resources by running multiple algorithms in a single MapReduce job. We execute multiple algorithms on the same underlying data in HDFS locality [8] (DataNode) at the same time(as shown in Figure 1). The proposed methodology can be applied on a wide range of algorithms (section V-A), however, for the simplicity and easiness of understanding, we explain it using WordCount" and "InvertedIndex" algorithms. For example, we have two algorithms "WordCount" [12] and "InvertedIndex" [13]. In a single map task the map function will be emitting (key, value) for each algorithm in the job (as shown in Figure 2) and

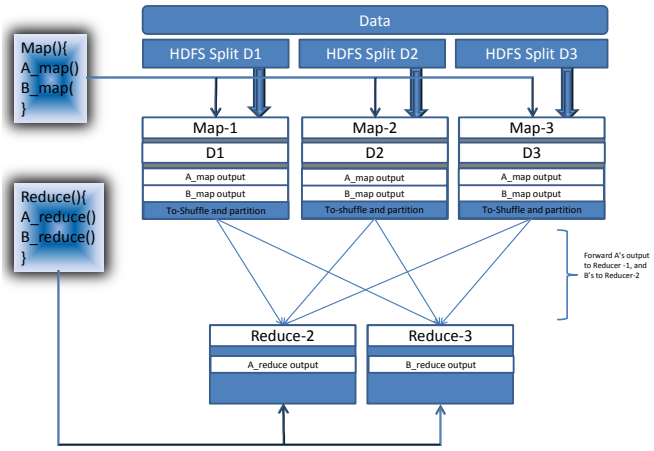

Fig. 1. In-Map/In-Reduce (architecture):In this figure, each mapper has two other map functions as *"A_map and B_map"* representing WordCount and InvertedIndex respectively, and reducers are customized to collect individual mapper's output.

the key structure of each algorithm is different. Different key structure is used in the reduce step to collect and distinguish the result of each algorithm.

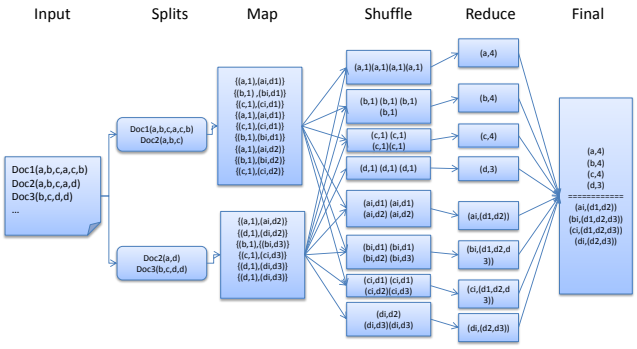

Fig. 2. In-Map/In-Reduce (example): In this figure, two mappers work on the data, each mapper emits (key,val) pairs for each algorithm(Word Count,Inverted Index) with different 'key' structures and this mapper output is shuffled, aggregated and final output is produced by he reducer.

The proposed methodology opens new ways of processing in the middle steps (aggregation, shufing and sorting) and will also help in mitigating skewed data of algorithms. In case of different algorithms as different jobs proposed in YARN [13], there is a resource manager that is necessary to divide the resources and schedulers to schedule each job individually and provide separate application manager to execute jobs other than MapReduce. The proposed method does not require separate schedulers, separate data and resources (data node, task trackers, and resource managers).

The rest of the paper is arranged as follows, Section II briey describes some of the related work in the eld of Big Data, HDFS and MapReduce. Section III presents our methodology by describing In-Map/In-Reduce in MapReduce in detail. Section IV presents the results, in section V we discuss and evaluate it, section VI concludes this paper.

II. Related Work

Currently many large organizations like Google [14], Yahoo [15], Oracle [16], and Facebook [17] are introducing new tools and technologies for fast and parallel processing of Big Data [1]. A new version of MapReduce (MR2.0 or YARN [13]) has been introduced by Apache Hadoop [2]. It has separated scheduling and resource management and introduced a global Resource Manager, per-application Application Master and per-node Node Manager. In this way it facilitates to run multiple applications alongside MapReduce.

In an effort to incorporate enterprise data into Big Data, Oracle has recently published a white paper[18], which integrates Hadoop, Oracle NoSQL and Oracle Data warehouse to provide Big Data solutions. In database community, solutions are provided to implement MapReduce on the structured data such as HadoopDB [19] and SCOPE [20].

For MapReduce many solutions has been presented by researchers for its task scheduling including LATE schedular [11] and SAMR Schedular[9]. These schedulers present a mechanism for task scheduling in MapReduce running in heterogeneous environments. They have also provided parallel execution of map jobs and reduce jobs i.e., when some of the map jobs are still running, execution of reduce jobs are started to collect data from mappers.

Oivos presented in [21] proposes a high-level declarative model and its run-time. It specifies the computation performed on a widely distributed and heterogeneous data sets. In MapReduce, some data sets that needs to be re-used and processed is performed by job chaining, while Oivos targets the data sets that are inter-dependent and programs to monitor, compile, and execute on these dependent data sets. In Oivos abstraction model, instead of writing separate programs for separate data sets, it writes programs in a single MapReduce program and then combines their and re-use their result in the same map function. Valvag et al. in [22] have proposed a system *Cogset*, which proposes tight-coupling between distributed systems. It introduces pre-defined communication and fault-tolerance mechanism calling it as *static routing*. It integrates storage file system with its execution environment i.e., tightly coupled. Oivos and Cogset both does not target same, homogeneous, and independent data sets. In-Map/In-Reduce executes different algorithm on the same and mutually independent data sets in a single MapReduce job.

In an attempt to introduce work-sharing across jobs in MapReduce, MRShare has been proposed by Nykiel and Potamias et all., [23]. MRShare enables automatic and principled work-sharing by transforming a batch of queries into a new batch that are executed efficiently by merging jobs into groups and evaluating each group as single query.

The above mentioned systems lack in providing parallel computation solution with parallel data processing. As the importance of MapReduce for Big Data is fairly understood, we believe that parallel execution of different algorithms in a single MapReduce job on the same underlying distributed data in HDFS will significantly improve the performance. Our solution decreases the time, cost and contention in MapReduce and it also decreases skew problem as defined and explained in [10].

III. METHODOLOGY

This section describes the methodology of our proposed approach, The process starts from the basic data upload step to the reduce step. The data is uploaded to HDFS to be used by algorithms e.g., WordCount as A and InvertedIndex as B (we use these two algorithms because of their simplicity). Map and Reduce phases are discussed in detail in the following sections 3.1 and 3.2 respectively.

A. In-Map/In-Reduce map function

In the map step, map function process input data in such a way that it generates two (key, value) pairs as (KA1, VA1) and (KB1, VB1) for A and B respectively. For *i* and *j* number of keys, it emits (KAi, VAi) and (KBj, VBj) key-value pairs for A and B respectively, in this way a single map function performs work of many maps as shown in algorithm 1.

Algorithm 1 In-Map/In-Reduce Map algorithm

1: **Input**: data split Dpi where D is the whole data, p is this data node local data and i the portion read for input to generate its pair e.g. (a word in a document in word count)
2: **Output**:(key, value) pair
3: **procedure** MAP(key, $value$)
4: A_map(key, value)
5: B_map(key, value)
6: **end procedure**

In Algorithm 1, we have used two functions A_map and B_map; their respective definitions are given in algorithm 2:

Algorithm 2 Individual algorithm Map functions

1: **Input**: key value pair, and some other data algorithm specific.
2: **Output**:(key, value) pair
3: **procedure** ALL_MAP(key, $value$)
4: split input and construct key
5: construct value
6: emit(key, $value$)
7: **end procedure**

a) : Algorithm 2 lists steps followed in map function for each algorithm in a job. After mappers complete, their generated (key,value) pairs are shuffled,aggregated and partitioned by the partitioner and collected by the reducers. Reduce function is described in detail in the next section.

B. In-Map/In-Reduce reduce function

After the map's output, based on pairs having same keys KAi and KBi i.e., KAi along with its all values aggregated are collected by a single reducer. The reducers collect values having same keys for each algorithm A and B from all mappers. The values along with unique keys are emitted as (KAi, SUM[VA1,.... VAk]) and (KBj, List [VB1,... VBk]) for algorithms A and B respectively as shown in algorithm 3.
In algorithm 3, we have used functions A_reduce and B_reduce, these are algorithm specific reduce functions and are given in algorithm 4:

Algorithm 3 In-Map/In-Reduce Reduce Algorithm

1: **Input**: key and a list of values
2: **Output**:(key, value) pair
3: **procedure** ALL_REDUCE(key, $value$)
4: A_reduce(key, $value$)
5: B_reduce(key, $value$)
6: **end procedure**

Algorithm 4 lists the steps followed in reduce function for each algorithm in a MapReduce job; these implementations can be different relating to algorithms. The reducer is designed in such a way that if it receives a key related to algorithm A, only that part of reducer is executed and the reducer's emit method is customized such that the output for algorithm A and B is stored in separate files. This way we can collect the result and then reuse it. Values belonging to the same key are collected by a reducer and this reducer is given in algorithm 3:

In the middle steps of the job i.e., shuffle, partition, and compare (sort) can also be implemented to dynamically manage the data partitioning, assigning to the reducers, and to avoid data skew and extra network communication. This paper scope does not cover skew mitigation, and aggregation.

C. In-Map/In-Reduce Custom Partitioner and Comparator

Besides map and reduce functions, the programmer would need to decide and design Partitioner and Comparator functions in MapReduce. In Partitioner, intermediate data is partitioned based on the key structure of each algorithm used in the map function. Partitioning based on key will combine and partition the data belonging to same algorithm together. Comparator is implemented in such a way that each algorithm's data is forwarded to a separate reducer or set of reducer depending on the data size. Custom partitioning helps in reducing data skew and load balancing between reducers. Custom comparator helps in separating the result of each algorithm based on the reducers. Implementation of custom Partitioner and Comparator significantly improve performance.

IV. RESULTS

We implemented two algorithms i.e., "WordCount" and "InvertedIndex" as separate MapReduce jobs as well as combined in a single MapReduce job (one job consisting of 2

Algorithm 4 Individual algorithm reduce functions

1: **Input**: key and a list of values
2: **Output**:(key, value) pair
3: **procedure** ALL_REDUCE(key,$List$)
4: $sum = 0$
5: $docsArray = 0$
6: For (each value in List) algorithm specific comp. $sum = sum + i$ e.g.,for wordcount $docsArray = docsArray + i^{\text{th}}$item e.g., for inverted index from the above, one is used in each function
7: emit(key, sum/List)
8: **end procedure**

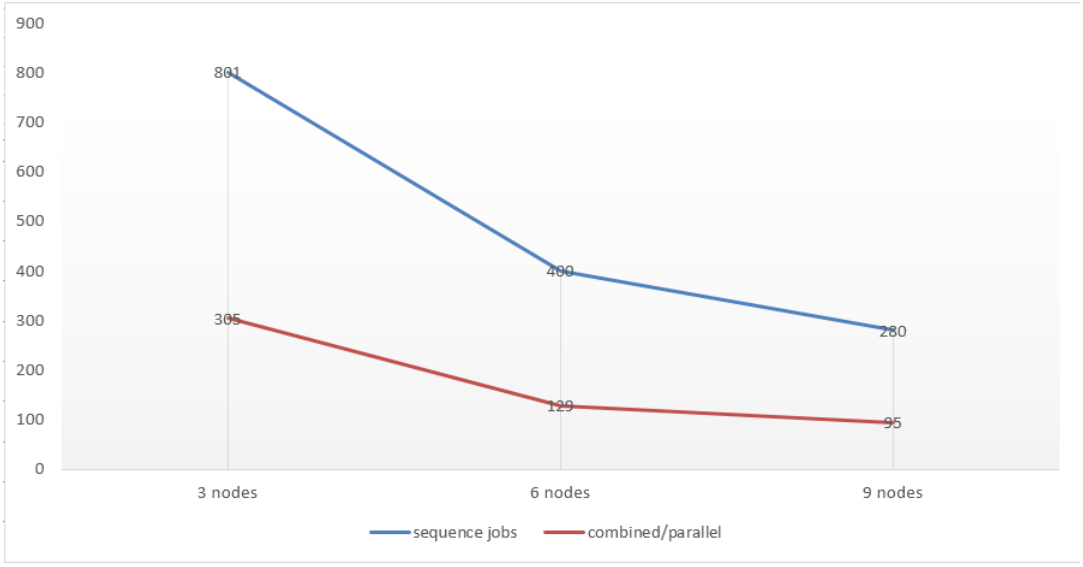

Fig. 3. comparison of MapReduce jobs in sequence and parallel execution. x-axis shows the number of nodes and y-axis shows time taken.

algorithms) as described in section 3. In separate jobs, we run one after another immediately after the completion of the first job (as chained jobs). In both cases, we use same data and execute both on varied cluster sizes (number of nodes). The data size used is more than 10GB from data-sets in [24] and [25]. The data nodes are having 4GB RAM, Intel CPU Core(TM) i5 3.4GHz and virtual machines on each physical node. Both cases are tested against each other to measure and compare their performance. Results are depicted in Figure 3. The comparison is shown in terms of the time taken to complete the job on different number of nodes in a cluster. It clearly indicates that both algorithms as separate jobs compared to parallel executions has significant difference and its time is reduced by 200% than chained jobs.

V. Discussion and Evaluation

In order to evaluate the performance of In-Map/In-Reduce, we compare it with MapReduce and notice the time cost as shown in results section. We estimate the cost complexity and it clearly indicates that its performance is 2 times better. Cost complexity is discussed in details below.

A. Cost Complexity

In generic MapReduce, the cost complexity [26] mainly depends on communication (between mappers and reducers to transfer the data) and computation costs [26]. Communication cost comprises of total input-output(I/O) of all processes (Elapsed communication cost counts the maximum of I/O along any path) and computation costs measures the running time of processes. So in MapReduce the total cost is:

$$Comm.cost = Msize + n(\sum i = 0...n(M^{i}proc.output)) + \sum j = 0...n(R^{j}output) \quad (1)$$

In equation 1, M is map process input size,n is the number of algorithms involved in MapReduce job, Mi represents output of each map process to be passed to reducers and Rj indicates each reducer process output. This equation tells the total communication cost of MapReduce algorithm and the elapsed communication cost is:

$$Elapsedcomm.cost = (M^{L}input + M^{L}outut) + (R^{L}input + R^{L}output) \quad (2)$$

In equation 2, L is the largest MR by size. Similarly computational cost is analogous, but we measure the running time of each process only. In generic MapReduce when job chaining is used, each job will have to read its input from HDFS and write back the output as shown in equation 3. *A* is the algorithm and *n* shows the number of algorithms involved.

$$Input - OutputCost(I/OCost) = (\sum i = 0...n(A^{i}loadinput)) + (\sum i = 0...n(A^{i}writeoutput)) \quad (3)$$

However, in In-Map/In-Reduce, this function is performed only once as the data is processed by all the algorithms in each mapper. Some extra cost is incurred in the shuffling phase, where each algorithm data from mapper's output is distributed to reducers. This is in-memory and much low cost as compared to I/O cost in equation 3.

Comparing costs in equation 1 and 2 with our methodology, the communication cost is reduced by (n-1)times the cost of n sequential MR jobs because we use M^{L} input and R^{L} output only once in single job for multiple algorithms. In generic MapReduce, M^{L} input and R^{L} output are separate for each algorithm running as single job in sequence or in separate jobs using unshared resources. In case of computational cost, our methodology is comparatively efficient because we perform the whole map tasks, then we move towards the reduce tasks. In case of generic MapReduce, the algorithms are separate jobs running in sequence or using unshared resources, therefore the comparison is a pair of largest MR process and many large MR processes in generic MapReduce i.e., max(A_map/B_map + A_reduce/B_reduce) . At the end either the communication cost dominates or the computation cost in MapReduce. In both cases parallel MR has less cost and mainly diverted from

communication cost to computation cost which really helps in reducing contention and network communication.

B. Applications and Tradeoffs

The proposed methodology can be used in variety of schemes including machine learning techniques, indexing techniques and multipurpose data extraction. Using this methodology, we can compare the results of various techniques and choose the suitable according to the demands of applications such as business analytics, visual analytics, and scientific data visualization. By combining multiple algorithms into a single MapReduce job, there are some tradeoffs that occur due to the nature of data and algorithms.

The performance of the system depends on the data locality, because when the map functions process the data in their local disks, they would not need to retrieve data from other replicas which results in decreasing the network communication. The processing of MapReduce becomes more compute intensive by the increase in computation of single mapper functions and it becomes less I/O bound by reading/writing the data only once for all algorithms. In In-Map/In-Reduce, when multiple algorithms are implemented to process the underlying data, the algorithms may need to be of the same nature. For example, when processing some data for clustering, we need to implement various clustering techniques and then use the results of a technique that best describes our problem. During the implementation of In-Map/In-Reduce, the programmers only need to provide the implementation of map/reduce functions and the key structure for each algorithm which will be used for the distinction in intermediate steps.

VI. Conclusion

In this paper, we have proposed concurrent job execution in MapReduce as In-Map/In-Reduce: A MapReduce framework, which supports multiple algorithm execution in a single MapReduce job on the same data in parallel. Experimental results have shown the effective performance of In-Map/In-Reduce. It decreases the execution time 2 times when only two algorithm are implemented and it increases with number of algorithms increasing and cluster size.

In-Map/In-Reduce is a part of our effort to introduce MapReduce as computation intensive platform along with its data intensive approach. It is a basic and initial step proposing parallel computational strategy in MapReduce, however further research is needed in the middle steps (aggregation, shuffling and partitioning) to handle skewness, scheduling and the resulting data in multi-algorithmic execution environment. The mappers output data can be used to mitigate skewness in a new fashion because it now involves many algorithms in a single job. In future, we are planning on extending this idea to improve performance in these areas and to cater the needs of efficient resource utilization in affordable cost.

VII. Acknowledgement

This research was supported by the MSIP(Ministry of Science, ICTFuture Planning), Korea, under the ITRC(Information Technology Research Center) support program supervised by the NIPA(National IT Industry Promotion Agency)" (NIPA-2013-(H0301-13-2001))

References

[1] D. Howe, M. Costanzo, P. Fey, T. Gojobori, L. Hannick, W. Hide, D. P. Hill, R. Kania, M. Schaeffer, S. St Pierre *et al.*, "Big data: The future of biocuration," *Nature*, vol. 455, no. 7209, pp. 47–50, 2008.

[2] T. White, *Hadoop: the definitive guide*. O'Reilly, 2012.

[3] J. Dean and S. Ghemawat, "Mapreduce: simplified data processing on large clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[4] K.-H. Lee, Y.-J. Lee, H. Choi, Y. D. Chung, and B. Moon, "Parallel data processing with mapreduce: a survey," *ACM SIGMOD Record*, vol. 40, no. 4, pp. 11–20, 2012.

[5] A. J. Hey, S. Tansley, K. M. Tolle *et al.*, "The fourth paradigm: data-intensive scientific discovery," 2009.

[6] A. Draganov, "Exploiting private and hybrid clouds for compute intensive web applications," 2011.

[7] P. Costa, A. Donnelly, A. Rowstron, and G. OShea, "Camdoop: Exploiting in-network aggregation for big data applications," in *USENIX NSDI*, vol. 12, 2012.

[8] Y. He, R. Lee, Y. Huai, Z. Shao, N. Jain, X. Zhang, and Z. Xu, "Rcfile: A fast and space-efficient data placement structure in mapreduce-based warehouse systems," in *Data Engineering (ICDE), 2011 IEEE 27th International Conference on*. IEEE, 2011, pp. 1199–1208.

[9] Q. Chen, D. Zhang, M. Guo, Q. Deng, and S. Guo, "Samr: A self-adaptive mapreduce scheduling algorithm in heterogeneous environment," in *Computer and Information Technology (CIT), 2010 IEEE 10th International Conference on*. IEEE, 2010, pp. 2736–2743.

[10] Y. Kwon, M. Balazinska, B. Howe, and J. Rolia, "Skewtune: mitigating skew in mapreduce applications," in *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*. ACM, 2012, pp. 25–36.

[11] A. MateiZaharia, A. Joseph, and I. RandyKatz, "Improving mapreduce performance in heterogeneous environments," 2010.

[12] G. Wu, "Wordcount example," 2011.

[13] A. Hadoop, "Apache hadoop nextgen mapreduce (yarn)," 2013.

[14] S. Ghemawat, H. Gobioff, and S.-T. Leung, "The google file system," in *ACM SIGOPS Operating Systems Review*, vol. 37, no. 5. ACM, 2003, pp. 29–43.

[15] R. T. Kaushik and M. Bhandarkar, "Greenhdfs: Towards an energy-conserving storage-efficient, hybrid hadoop compute cluster," in *Proceedings of the USENIX Annual Technical Conference*, 2010.

[16] S. Shankar, A. Choi, and J.-P. Dijcks, "Integrating hadoop data with oracle parallel processing–an oracle white paper," *Oracle Corporation*, 2010.

[17] D. Borthakur, "Facebook has the worlds largest hadoop cluster," *Retrieved April*, vol. 20, p. 2012, 2010.

[18] J. P. Dijcks, "Oracle: Big data for the enterprise," *Oracle White Paper*, 2012.

[19] J. Zhou, N. Bruno, M.-C. Wu, P.-A. Larson, R. Chaiken, and D. Shakib, "Scope: parallel databases meet mapreduce," *The VLDB Journal-The International Journal on Very Large Data Bases*, vol. 21, no. 5, pp. 611–636, 2012.

[20] H. Kllapi, D. Bilidas, I. Horrocks, Y. Ioannidis, E. Jimenez-Ruiz, E. Kharlamov, M. Koubarakis, and D. Zheleznyakov, "Distributed query processing on the cloud: the optique point of view (short paper)," in *OWL Experiences and Directions Workshop (OWLED)*, 2013.

[21] S. V. Valvag and D. Johansen, "Oivos: Simple and efficient distributed data processing," in *High Performance Computing and Communications, 2008. HPCC'08. 10th IEEE International Conference on*. IEEE, 2008, pp. 113–122.

[22] ——, "Cogset: A unified engine for reliable storage and parallel processing," in *Network and Parallel Computing, 2009. NPC'09. Sixth IFIP International Conference on*. IEEE, 2009, pp. 174–181.

[23] T. Nykiel, M. Potamias, C. Mishra, G. Kollios, and N. Koudas, "Mrshare: sharing across multiple queries in mapreduce," *Proceedings of the VLDB Endowment*, vol. 3, no. 1-2, pp. 494–505, 2010.

[24] C. for Machine Learning and I. Systems, "Machine learning data-sets: http://archive.ics.uci.edu/ml/datasets.html," Last seen: May 30th, 2014.

[25] Rdatamining.com, "R and data mining data-set: http://www.rdatamining.com/resources/data," Last seen: May 30th, 2014.

[26] S. University, "Mapreduce computation," 2009.