Flying Yellow Elephant: Predictable and Efficient MapReduce in the Cloud

Jörg Schad

Supervised by: Prof. Dr. Jens Dittrich

Information Systems Group, Saarland University
http://infosys.cs.uni-saarland.de

ABSTRACT

Today, growing datasets require new technologies as standard technologies — such as parallel DBMSs — do not easily scale to such level. On the one side, there is the MapReduce paradigm allowing non-expert users to easily define large distributed jobs. On the other side, there is Cloud Computing providing a pay-as-you-go infrastructure for such computations. This PhD project aims at improving the combination of both technologies, especially for the following issues: (i) predictability of performance, (ii) runtime optimization and (iii) Cloud-aware scheduling. These issues can result in significant runtime overhead or non-optimal use of computing resources, which in a Cloud setting directly correlates to high monetary cost. We present preliminary results that confirm a significant improvement on performance when addressing some of these issues. Further, we discuss research challenges and initial ideas for above mentioned issues.

1. INTRODUCTION

We currently face an enourmous growth of datasets — up to PetaBytes — including customer data, Web logs, web indexes, sales data and many more. As datasets grow, the difficulty of analyzing such datasets increases as well, because parallel DBMSs require massive effort to scale up to such level. Hence, new types of frameworks for data analysis scaling to this level are required. Technologies as MapReduce and Cloud Computing are emerging to deal with this.

On the one side, the MapReduce framework, introduced by Google in 2004 [19], aims at dealing with large scale datasets by utilizing large commodity hardware clusters. Users only have to provide two User Defined Functions (UDFs) — map and reduce — and do not have to worry about parallelization, fault-tolerance, or tuning several parameters of the system. The main motivation of developing the MapReduce framework was to create the billion-page web index. Today, MapReduce is used for 80% of the data analysis at Google including machine learning problems, usage reports, and even processing of satellite image data [41]. In particular, with the Apache open source framework Hadoop [4] the usage

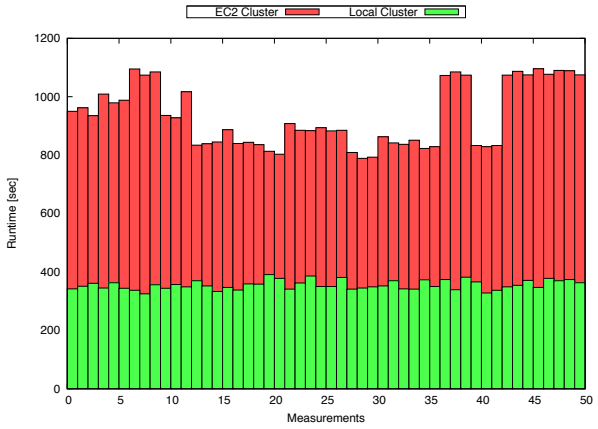

Figure 1: MapReduce Performance on the Cloud compared to a Local Cluster

of MapReduce has been spread to a large number of other applications as well. MapReduce is currently used by a growing number of companies like Yahoo!, Facebook, AOL, A9, Last.fm, and The New York Times [5].

On the other side, Cloud computing is a model that allows users to easily access and configure a large pool of remote computing resources (i.e. a *Cloud*). This model has gained a lot of popularity mainly due to its ease of use and its ability to scale up on demand. As a result, several providers such as Amazon, IBM, Microsoft, and Yahoo already offer this technology. For many users, especially for researchers and medium-sized enterprises, the cloud computing model is quite attractive because it is up to cloud providers to maintain the hardware infrastructure.

These two technologies — especially in combination — make large-scale data analysis feasible even for non-expert users: while MapReduce allows non-expert users to perform complex tasks over large datasets, Cloud Computing provides the necessary infrastructure for running such large-scale applications. For example, the New York Times used Hadoop and Amazon EC2 to convert over $400,000$ large TIFF images and additional metadata of the TimeMachine [10] — which is an archive of full-page scans of New York Times issues from 1851–1922 — to smaller web ready representation as well as to generate the Java Script code needed for the mouse-over effects in under 36 hours.

1.1 Motivation

For many users, especially for researchers and medium-sized en-

.

terprises, Cloud computing model is quite attractive because it is up to the cloud providers to maintain the hardware infrastructure. Despite the attention paid by cloud providers, some cloud computing nodes may attain orders of magnitude worse performance than other nodes [13]. This indeed may considerably influence performance of MapReduce jobs, which are usually long-running jobs and thus more sensible to hardware performance variations. For example, we show the runtimes of a MapReduce job for a 50-node EC2 cluster and a 50-node local cluster in Figure 1. We can easily see that performance on EC2 varies considerably.

Performance unpredictability in the cloud is in fact a major issue for many users and it is considered as one of the major obstacles for cloud computing [13]. For example, researchers expect comparable performance for their applications at any time, independent of the current workload of the cloud; this is quite important for researchers because of repeatability of results. Another example are enterprises that depend on *Service Level Agreements*. A way to deal with such a performance unpredictability is to modify the execution plans of jobs according to the current performance of the Cloud. Nevertheless, MapReduce uses the same physical query plan for processing any job. This usually yields to inefficient query plans for MapReduce jobs and does not allow for any change in the plan. This usually results in long runtimes that directly correlates to high monetary cost in public Clouds.

2. PROBLEM STATEMENT

MapReduce allows non-expert users to perform large-scale data processing, while Cloud computing offers the required infrastructure for such analysis. This is already commonly done, e.g. Amazon Elastic Map Reduce Service. However, there are several open issues — such as performance variance on the cloud and static scheduling of MapReduce jobs — that are not addressed yet and have a negative impact on performance of applications.

Hence, the problem we consider is how to efficiently execute MapReduce jobs in a Cloud computing setting.

We discuss the research challenges we face when dealing with above problem in the next section and discuss related work in Section 4. We then discuss and present our first prototype to solve above problem and our preliminary results in Section 5. Finally, we conclude this paper in Section 6.

3. RESEARCH CHALLENGES

3.1 Predictability of MapReduce Jobs on the Cloud

If the Cloud will serve as the underlying platform for performing large scale analysis, one has to deal with a number of issues not present in a controlled local environment: High runtime variance, node failures, straggling nodes, and varying availability. For example the runtime variance is shown in Figure 1 where the same MapReduce job is executed on a 50 node virtual EC2 cluster and on a 50 virtual node local cluster. The observed cloud variance makes experimental results hard to interpret and compare. Also, predictability of runtimes is virtually not possible which limits the usage of Cloud Computing for a number of applications such as scientific benchmarking.

3.2 Runtime Optimizations of MapReduce

The current implementation of Hadoop uses a hard-coded execution plan often yielding overhead. For example, data access is always performed in a scan-like fashion and cannot use other access paths such as indexes. A recent study [33] showed that shared-nothing DBMSs outperform standard MapReduce by a large factor in a variety of tasks. The key for DBMSs to achieve such performance is a flexible physical query plan that can exploit metadata, such as data schema or distribution. On the other hand, MapReduce provides a simple interface allowing non-expert users to run complex jobs in a scalable manner, which is not the case for standard DBMSs. However, in a large pay-as-you-go cluster runtime easily amounts to considerable monetary cost.

Therefore, the challenge is to make the performance of MapReduce comparable to shared-nothing DBMSs without changing the MapReduce programming paradigm. But also, we should make only minor changes to the MapReduce execution framework so as to keep it compatible with future implementation versions. Additionally, another aspect to consider is the execution of concurrent jobs — a setting frequently occurring in Cloud Computing —, because they may compete for computing resources and input data.

3.3 Cloud-Aware MapReduce Scheduling

One key feature of Cloud Computing is flexibility; users can easily adjust the required computing resources on the fly and for example aquire additional nodes. Also, there exist usually a variety of different instance types having different CPU, IO, or Memory performance and pricing [23]. This feature yields a number of opportunities for a MapReduce scheduler which normally is dealing with mostly static clusters:

1. **Optimal Cloud Size**. In contrast to a normal cluster the number of nodes in Cloud is not fixed and could even be changed during job runtime. As more nodes usually result in higher monetary cost, an interesting application is to obtain an optimal setting constraint by some monetary cost. For instance, a non-expert user desiring to run a MapReduce job within 2 hours would need the scheduler to automatically assign the number of node instances to run the job.

2. **Heterogeneous Cloud**. As the different instance types deliver different performance — often vendors offer special high I/O performance or high CPU instance nodes — at different prices, a given MapReduce cluster might consist of different node types which need to be optimally used. The goal again would be to achieve the best performance/price ratio. For example, I/O intensive tasks might be better scheduled to nodes instances having high I/O performance, while CPU intensive tasks might be scheduled to high-CPU-performance node instances.

3. **Dynamic Scheduling**. As with Cloud Computing the above mentioned points also may change during runtime of a job, the scheduler needs to be dynamically react to such changes — such as reacting to change in the workload or even to failing nodes.

4. **Cost-awareness**. As in public Clouds users are charged for the computing resources they use, the scheduler should consider monetary-cost constraints given by users. For example, one user may desire to run one MapReduce job paying at most $100 and receiving the best possible performance for this price.

4. RELATED WORK

4.1 MapReduce

Over the past three years MapReduce has attained considerable interest from both the database and systems research community [15, 27, 38, 31, 34, 25, 11, 36, 16, 37, 29, 17]. As a result, some DBMS vendors have started to integrate MapReduce front-ends into their systems including Aster, Greenplum, and Vertica. However, these systems do not change the underlying execution system: they simply provide a MapReduce front-end to a DBMS. Recently, [11] proposed HadoopDB, a new system that combines techniques from DBMSs, Hive [36], and Hadoop. In summary, HadoopDB can be viewed as a data distribution framework to combine local DBMSs to form a *shared-nothing DBMS*. The results in [11] however show that HadoopDB improves task processing times of Hadoop by a large factor to match the ones of a shared-nothing DBMS. Nevertheless, above systems are still databases. Therefore, in contrast to MapReduce, they require advanced knowledge from the user-side on database management, data models, schemas, SQL, load distribution and partitioning, failure handling, and query processing in general, but also on the specific product in particular. On the other side, much work has been done on scheduling MapReduce jobs with different goals in mind. Hadoop for example include a Fair and Capacity scheduler with the aim of sharing computing cluster among jobs [4]. However, the homogeneity assumption made by Hadoop might lead to a degradation on performance in heterogenous clusters such as the Cloud. Zaharia et al. [40] proposed a scheduler to schedule MapReduce jobs in heterogeneous MapReduce clusters. Nonetheless, this work does not allow users to neither dynamically change the setup of their experiments nor take into account monetary costs of jobs. Deshpande et al. considered the aspect of adaptive query processing in traditional Database Systems ([20]), which also yields interesting aspects for MapReduce; especially with long running queries on large datasets it becomes important to adaptivly tune the query.

4.2 Cloud Computing

Cloud computing has been the focus of several research works and is still gaining more attention from the research community. There exist a number of research works on testbeds for cloud computing. For example, the Open Cloud Consortium [6] and OpenCirrus [14] aim at developing a cloud computing infrastructure, targeting the research community in particular. Also, some grid testbeds started to make efforts to fuse grids with clouds [7, 9]. Other efforts such as Eucalyptus [30] and Tashi [8] aim at providing software infrastructure for implementing and managing cloud computing on clusters. However, none of these works focuses on the particular issues raised by cloud computing mentioned earlier.

Cryans et al. [18] compare the cloud computing technology with database systems and propose a list of comparison dimensions. Garfinkel [24] evaluates the different cloud services of Amazon in terms of cost and performance, but he does not provide any evaluation of the possible impact that performance variance may have on users applications. Finally, new projects that monitor the performance of clouds have recently emerged. For example, CloudClimate [1] and CloudKick [2] already perform performance monitoring of different clouds. EC2 also offers CloudWatch [3] which provides monitoring for *Amazon Web Services* cloud resources. However, none of the above works focuses on evaluating the possible performance variability in clouds or even give hints on how to reduce this variability.

Unfortunatly there is little research concerning the intersection of Cloud Computing and MapReduce or the particular challenges arising there, which we briefly discuss in the following section.

Besides the mentioned works there exist a number of other papers ([28, 26]) in the field of distributed query processing having relevance to this topic, but most of them do not consider (a) the specific challenges of MapReduce and (b) the flexible and changing Cloud environment.

5. PRELIMINARY RESULTS

In order to solve the problem of efficiently uniting MapReduce and Cloud Computing we propose the following approaches.

5.1 Predictability of MapReduce Jobs on the Cloud

Both Cloud Computing and MapReduce are two technologies that have gained a lot of popularity mainly due to its ease-of-use and its ability to scale up on demand. As a result, MapReduce jobs are a popular application on the Cloud. For many users — especially for researchers and medium-sized enterprises — the Cloud Computing model is quite attractive because it is up to cloud providers to maintain the hardware infrastructure. However, despite the attention paid by Cloud providers, some nodes may attain orders of magnitude worse performance than other nodes [35]. This indeed may considerably influence the performance of their MapReduce jobs.

Unfortunately, it is currently not possible to deal with this issue from the application layer. Users should therefore conduct wall clock experiments with considerable care. With this in mind, we believe that applications can be variance-aware so as to produce more meaningful results, i.e. normalize results according to the variance experienced by applications.

Preliminary Results. We evaluated the performance variance of popular cloud provider EC2 [23] since we experienced large performance variance when running large MapReduce jobs 50 node cluster on the cloud (see Figure 1). More detailed experiments showed that this MapReduce variance arises from variance at the different components. For example, the CPU performance variance of a series of measurements is shown in Figure 2. But we could also show that a large part of this variance arises from differences in the underlying architecture and could be reduced when considering those differences [35]. But even when considering those differences, the performance varies significantly due to contention by different virtual machines. We also gave hints on how to reduce the runtime variance. As future work we plan to inject several performance micro benchmarks into MapReduce so as to normalize results and make them comparable. Those micro benchmarks should be invisible to user and not change MapReduce system. Therefore we plan to introduce so called *Trojan Benchmark* by overwritting or wrapping one of the user defined functions.

5.2 Runtime Optimizations of MapReduce

When optimizing the performance of MapReduce jobs, one faces a number of manual decisions, which are not trivial for users to make. For this reason, systems like Pig [32] and Hive [36] include a query optimizer in their query language. However, these systems have several drawbacks. First, they use ruled-based optimizations, which essentially only consider basic optimizations such as predicates pushdown, partition pruning, and map-side joins. Second, they simply translate queries (specified in their own query language) to MapReduce jobs, which are finally performed by MapReduce framework using the same query physical plan yielding to inefficient query plans for several queries. Third, as they only compile queries into MapReduce jobs, they cannot take into consider-

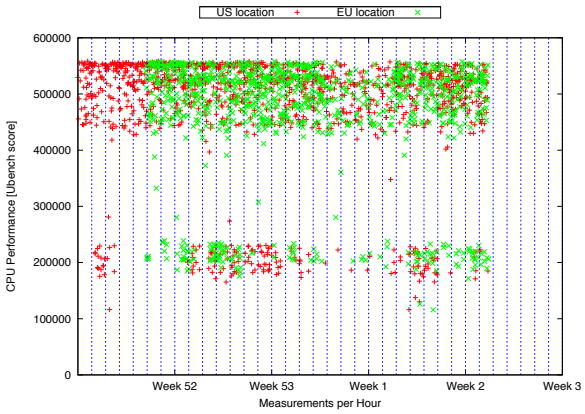

Figure 2: CPU performance on large EC2 instances.

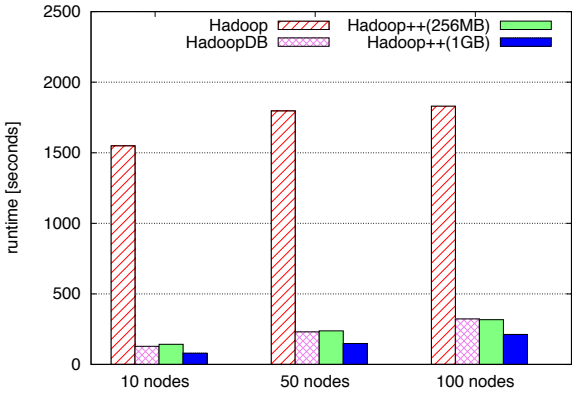

Figure 5: Runtime Improvements related to Indexing and Join Processing

ation dynamic settings such as current load of nodes. Finally, they force users to use SQL-like query interfaces, which is one of the reasons why users move from DBMSs to MapReduce.

Even though, Hadoop has a fixed query plan (which does not always allow for the optimal execution plan [21, 12]) it still allows the user to specify user defined functions. Using Hadoop users can not only specify the two user defined UDFs -Map and Reduce- as one would expect but actually one can specify up to 10 different UDFs concerned with reading data, sorting and more. One example could be the use of an index instead of an entire file by changing the InputFileFormat. In order to make this decisions without changin the Hadoop framework or involving the user we need to "inject" the functionality into the framework.

One example of such "injection" is the *Trojan Index*, which is our solution to integrate indexing capability into Hadoop. The salient features of this approach include being non-invasive and providing optional index access path.

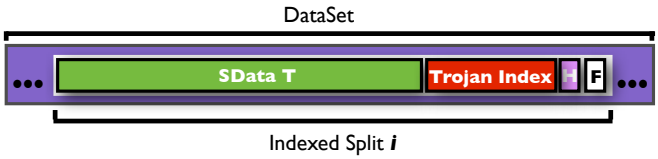

Figure 3: Trojan Index Data Layout.

We illustrate the core idea of Trojan Index in Figure 3. For each split of data (SData T) a covering index (Trojan Index) is built. Additionally, a header (H) is added. It contains indexed data size, index size, first key, last key and number of records. Finally, a split footer (F) is used to identify the split boundary.

Join processing is another area where we can apply the idea of trojan techniques. Currently in MapReduce, two datasets are joined using *re-partitioning*: partitioning records by join key in the map phase and grouping records with the same key in the reduce phase. The reducer joins the records in each key-based group.

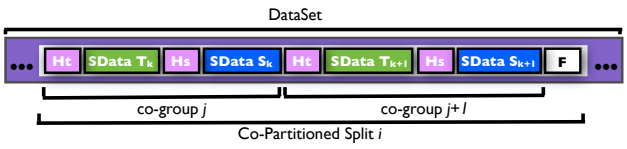

Figure 4: Co-partitioned Data Layout

Trojan Join is our solution to support more effective join processing in Hadoop. The core idea is the use of schema information to *co-partition* the data at load time — i.e. given two input relations, we apply the same partitioning function on the join attributes of both the relations at data loading time — and place the *co-group* pairs, having the same join key from the two relations, on the same node. The resulting data layout is shown in Figure 4. Trojan joins are now processed locally within each node at query time and hence reduce the network traffic significantly.

Preliminary Results. By introducing simple non-invasive new index and join techniques, we have seen that simple optimizations can significantly improve runtimes of MapReduce jobs without changing the underlying framework. Therefore we are building a system coined Hadoop++ [22]. We compared it against the original Hadoop and HadoopDB [11] using the benchmark as defined in [34] on Amazons EC2 [23]. The performance of the join task is shown in Figure 5. We observe that Hadoop++ performs about a factor 20 better than the original Hadoop and is comparable to HadoopDB, which utilizes the indexes and optimizations of the underlying DBMSs. In contrast to HadoopDB, Hadoop++ does not require underlying Database Systems and does not change the standard MapReduce interface to SQL. As future work we plan to introduce more flexibility into Hadoop to allow for more complex optimizations.

5.3 Cloud-Aware MapReduce Scheduler

In addition to the usual mapping of tasks to individual nodes, the Scheduler in the Cloud setting will have to also consider the optimal cluster configuration with respect to number of nodes and different instance types. We also integrate the heterogeneity resulting from the different instance types when scheduling tasks.

In the beginning, we consider this simply an optimization problem with given constraints such as maximal cost or runtime. The Scheduler will be a deeper change to the Hadoop framework, but as the Scheduler is a pluggable component [39] we do not need to change core Hadoop code.

6. CONCLUSION

In this paper, we discussed several challenges arising when executing MapReduce jobs on the Cloud. We showed that there exist several open problems caused by the unpredictability of Cloud performance. But also, we showed that there are a number of opportunities for improving MapReduce jobs performance on the Cloud, such as Cloud-aware scheduling. As users of public Clouds are charged for the computing resources they use, runtime optimiza-

tions are crucial to decrease cost for users. We strongly believe one can significantly improve MapReduce performance on the Cloud. The preliminary results of our first efforts confirm in fact a significant improvement on performance when taking into account some of the ideas we discuss in this paper such as Trojan Indexes and Trojan Joins techniques. But again, further optimizations are still possible. This motivates us to carry on our research efforts on this trend with the aim of efficiently executing MapReduce jobs in a Cloud computing setting.

7. REFERENCES

[1] CloudClimate, http://www.cloudclimate.com.

[2] CloudKick, https://www.cloudkick.com.

[3] CloudWatch, http://aws.amazon.com/cloudwatch.

[4] Hadoop, http://hadoop.apache.org/mapreduce/.

[5] Hadoop users, http://wiki.apache.org/hadoop/poweredby.

[6] Open Cloud Consortium, http://opencloudconsortium.org.

[7] PlanetLab, http://www.planet-lab.org.

[8] Tashi project, http://incubator.apache.org/tashi.

[9] TeraGrid, http://www.teragrid.org.

[10] Timesmachine, http://timesmachine.nytimes.com.

[11] A. Abouzeid, K. Bajda-Pawlikowski, D. Abadi, A. Rasin, and A. Silberschatz. HadoopDB: An Architectural Hybrid of MapReduce and DBMS Technologies for Analytical Workloads. In *PVLDB*, 2009.

[12] F. Afrati and J. Ullman. Optimizing Joins in a Map-Reduce Environment. In *EDBT*, 2010.

[13] M. Armbrust, A. Fox, R. Griffith, A. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, et al. Above the Clouds: A Berkeley View of Cloud Computing. *EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2009-28*, 2009.

[14] R. Campbell, I. Gupta, M. Heath, S. Ko, M. Kozuch, M. Kunze, T. Kwan, K. Lai, H. Lee, M. Lyons, et al. Open CirrusTM Cloud Computing Testbed: Federated Data Centers for Open Source Systems and Services Research. In *USENIX Workshop on Hot Topics in Cloud Computing*, 2009.

[15] R. Chaiken, B. Jenkins, P.-Å. Larson, B. Ramsey, D. Shakib, S. Weaver, and J. Zhou. Scope: Easy and Efficient Parallel Processing of Massive Data Sets. In *PVLDB*, 2008.

[16] J. Cohen, B. Dolan, M. Dunlap, J. Hellerstein, and C. Welton. Mad Skills: New Analysis Practices for Big Data. In *PVLDB*, 2009.

[17] T. Condie, N. Conway, P. Alvaro, J. Hellerstein, K. Elmeleegy, and R. Sears. MapReduce Online. In *USENIX NSDI*, 2010.

[18] J.-D. Cryans, A. April, and A. Abran. Criteria to Compare Cloud Computing with Current Database Technology. In *Conf. on Software Process and Product Measurement*, 2008.

[19] J. Dean and S. Ghemawat. Mapreduce: Simplified Data Processing on Large Clusters. In *OSDI*, 2004.

[20] A. Deshpande, Z. Ives, and V. Raman. Adaptive Query Processing. *Foundations and Trends in Databases*, 1(1):1–140, 2007.

[21] D. DeWitt, E. Paulson, E. Robinson, J. Naughton, J. Royalty, S. Shankar, and A. Krioukov. Clustera: an Integrated Computation and Data Management System. In *PVLDB*, 2008.

[22] J. Dittrich, J.-A. Quiané-Ruiz, A. Jindal, V. S. Yagiz Kargin, and J. Schad. Hadoop++: Making a Yellow Elephant Run Like a Cheetah (Without It Even Noticing). In *PVLDB*, 2010.

[23] Amazon Elastic Compute Cloud. http://aws.amazon.com/ec2/.

[24] S. Garfinkel. An Evaluation of Amazon's Grid Computing Services: EC2, S3 and SQS. Technical Report TR-08-07, Harvard University, July 2007.

[25] A. Gates, O. Natkovich, S. Chopra, P. Kamath, S. Narayanam, C. Olston, B. Reed, S. Srinivasan, and U. Srivastava. Building a HighLevel Dataflow System on Top of MapReduce: The Pig Experience. 2009.

[26] A. Gounaris, R. Sakellariou, N. Paton, and A. Fernandes. A Novel Approach to Resource Scheduling for Parallel Query Processing on Computational Grids. *Distributed and Parallel Databases*, 19(2):87–106, 2006.

[27] M. Isard, M. Budiu, Y. Yu, A. Birrell, and D. Fetterly. Dryad: Distributed Data-Parallel Programs from Sequential Building Blocks. In *EuroSys*, 2007.

[28] D. Kossmann. The State of the Art in Distributed Query Processing. *ACM Comput. Surv.*, 32(4):422–469, 2000.

[29] K. Morton and A. Friesen. KAMD: A Progress Estimator for MapReduce Pipelines. In *ICDE*, 2010.

[30] D. Nurmi, R. Wolski, C. Grzegorczyk, G. Obertelli, S. Soman, L. Youseff, and D. Zagorodnov. The Eucalyptus Open-Source Cloud-Computing System. In *IEEE/ACM International Symposium on Cluster Computing and the Grid*. IEEE Computer Society, 2009.

[31] C. Olston, B. Reed, U. Srivastava, R. Kumar, and A. Tomkins. Pig Latin: A Not-So-Foreign Language for Data Processing. In *SIGMOD*, 2008.

[32] C. Olston, B. Reed, U. Srivastava, R. Kumar, and A. Tomkins. Pig Latin: A Not-So-Foreign Language for Data Processing. In *SIGMOD*, 2008.

[33] A. Pavlo, E. Paulson, A. Rasin, D. J. Abadi, D. J. DeWitt, S. Madden, and M. Stonebraker. A comparison of approaches to large-scale data analysis. In *SIGMOD*, 2009.

[34] A. Pavlo, E. Paulson, A. Rasin, D. J. Abadi, D. J. DeWitt, S. Madden, and M. Stonebraker. A comparison of approaches to large-scale data analysis. In *SIGMOD*, 2009.

[35] J. Schad, J. Dittrich, and J.-A. Quiané-Ruiz. Runtime Measurements in the Cloud: Observing, Analyzing, and Reducing Variance. In *PVLDB*, 2010.

[36] A. Thusoo, J. S. Sarma, N. Jain, Z. Shao, P. Chakka, S. Anthony, H. Liu, P. Wyckoff, and R. Murthy. Hive - A Warehousing Solution Over a Map-Reduce Framework. In *PVLDB*, 2009.

[37] C. Yang, C. Yen, C. Tan, and S. Madden. Osprey: Implementing MapReduce-Style Fault Tolerance in a Shared-Nothing Distributed Database. In *ICDE*, 2010.

[38] H. Yang, A. Dasdan, R. Hsiao, and S. Parker. Map-Reduce-Merge: Simplified Relational Data Processing on Large Clusters. In *SIGMOD*, 2007.

[39] M. Zaharia, D. Borthakur, J. Sarma, K. Elmeleegy, S. Shenker, and I. Stoica. Job Scheduling for Multi-User Mapreduce Clusters. *EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2009-55, Apr*, pages 2009–55, 2009.

[40] M. Zaharia, A. Konwinski, A. D. Joseph, R. H. Katz, and I. Stoica. Improving MapReduce Performance in Heterogeneous Environments. In *OSDI*, 2008.

[41] J. Zhao and J. Pjesivac-Grbovic. MapReduce: The Programming Model and Practice. In *SIGMETRICS*, 2009.