Resource Based Assignment of Queries in Distributed Databases

David Dietrich, Tzu-Yang Yu, Yiyao Liu
David R. Cheriton School of Computer Science
University of Waterloo
Waterloo, Canada
{d4dietri, t32yu, y435liu}@uwaterloo.ca

ABSTRACT

Distributed database is blablabla. However, a disadvantage of such database is the imbalanced workload assigned to each node server and thus the inefficiency of the whole system. In this paper, we examine how the different CPU and memory usage on a machine affect the query execution time. Based on that, we try to assign jobs to different node servers according to their conditions. From our experiment,

1. INTRODUCTION

Recently, the cloud has attracted a lot of attention from both the scientific research community and practitioners as it enables executing long running resource intensive tasks on the cloud using as many nodes as required thus greatly reducing the task completion time and eliminating the need to procure expensive server grade hardware [2]. However, the performance of cloud compute nodes is often not consistent, with some nodes obtaining orders of magnitude worse performance than other nodes. There are a variety of reasons a node could suffer from degraded performance, ranging from extremely heavy workload (i.e., the node is a hotspot), to partial hardware failure [1]. However, the modern scheduling systems only simply assigned query to the closest replica [3], without taking heterogeneous hardware into account. This eventually limits communication between servers, and making congestion and computation hot-spots prevalent even when spare capacity is available elsewhere.

2. RELATED WORK

To work around this problem, many researchers focus on estimating workload resource usage in order to maximal performance while minimizing the cost of resources used [12, 8, 6, 13]. This predict job execution time and resource requirements technique can help cloud provider to make decision about which requests from which users are to be executed on which computational resources, and when [4]. Many researchers had built workload placement recommendation service base on workload demand patterns [20,21], and such approach can result in 35% reduction in processor usage [20]. Differ to our study, our system do not take workload requirement perdition into account, because we believe that execute workload analysis for every query will result in some latency issues. Instead, our system focuses on dynamically forwarding workload to the idlest server, in order to prevent hotspot.

Furthermore, some other researchers focus on partitioning schemes. Works like graph partitioning algorithms [8], workload-aware partitioning [9], and classical work on physical design and partitioning [10] are focusing on dividing data into partitions that maximize transaction/query performance to allow workloads to scale across multiple computing nodes. However, graph partitioning is NP-hard problems, and solutions to this problem are generally derived using heuristics and approximation algorithms [22], and we believe that this will introduce execution complexity, and the performance is vary to different databases. Next, the workload-aware partitioning focus on partition tables base on workload prediction [9, 23]; however, there are many workloads, a ïňAnite number can be hosted by each server, and each workload has capacity requirements that may frequently change based on business needs [20], this means that workload-aware partitioning needs to be dynamic rather than static; however dynamic workload-aware partitioning will reduce performance dramatically. Last, some other works on physical design and partitioning allows database loaded into the system by both static decision and dynamic decision in order to increase I/O bandwidth [10]. This approach is pragmatic, but if we only rely on partitioning database the performance will be limited.

Virtual machine (VM) migration is also another approach for load balancing. VM migration focuses on the transfer of a VM from one physical machine to another with little or no service downtime (e.g. live VM migration) when the server that the VM sits on becomes overloaded with processes, traffic, or memory usage [11]. Nevertheless, the challenges with long distance live migration are the WAN bandwidth, latency, and packet loss limitations that are outside the control of most IT organizations. Many applications are susceptible to network issues across the WAN that can be exacerbated by distance and network quality [25], in addition, traditional VM migration will cause network traffic and bandwidth consumption, and many VM migration researcher are struggling in mitigating those causes [24].

3. DESIGN
4. IMPLEMENTATION
5. EVALUATION

To evaluate our scheduling algorithm we are comparing the performance of Apache Cassandra (version 1.1.6) [10] with and without our scheduler. We have chosen to use Cassandra because it is an open-source key-value database with a large community and is used by several enterprise clients [7]. To evaluate the performance of each version of Cassandra we are using the Yahoo! Cloud Service Benchmark (YCSB) [5]. The remainder of this section will describe the experimental setup and the results of the experiments.

5.1 Experimental Configuration

Our experiments were performed on a 10-server cluster running Ubuntu Linux 12.04. Each server has a 6-core 2.3GHz processor and 16GB of main memory. The machines are all located in close proximity (latency did not prove to be an issue in the experiments).

The data is Cassandra was partitioned using the *RandomPartitioner* (equivalent to Consistent Hash Partitioning [11]). The data was therefore equally distributed between all 10 nodes in the cluster (the Master server contained data to be queried on, but this did not appear to result in the master being overloaded). The replication factor used in our database was 3. The read consistency was set to *one* (meaning that the read query only looks at a single replica to get a result).

The default method for assigning queries to nodes in a Cassandra cluster is the *SimpleStrategy*. This method just assigns the query to the first replica in the ring. It should be noted that this method is only adequate for single datacenter databases (which is one of our assumptions). Cassandra offers other choices (e.g., a method for a database distributed over multiple data centers and for use with AmazonâĂŹs EC2 cloud environment), but given our experimental hardware configuration we did not explore these methods.

YCSB is used to generate and insert the data for the experiments. All of the data is stored in a single database table. There are 10 values associated with each key, with each value being 100 bytes. We ran our experiments on databases with 10 million keys and 150 million keys (corresponding to 10GB and 150GB databases). This was done to examine the impact that CassandraâĂŹs key and row caching would have on the comparative performance of the versions of Cassandra.

We ran two different kinds of workloads against Cassandra: a 100% read workload, and a 100% scan (multi-key read) workload. The read-heavy workload was used because reads are the most time consuming single-key operation, and likewise scans are the most time consuming operation overall. The maximum scan length was set to 1000 keys, with the distribution over that range being uniform.

The queries are generated dynamically conforming to a Zipfian distribution [9] with a skew of 0.99. We chose to use a Zipfian distribution so that the workload would differ between machines in the cluster (thus making our scheduling algorithm more applicable). The YCSB clients were run from one of the machines in the cluster (not the master). This did not appear to have any effect on the performance of the scheduler.

The performance is measured in terms of operations per second that are performed. This is the default method that YCSB provides to measure performance. The YCSB also offers the ability to set a target for the number of operations that should be performed per second (similar to deadlines on the queries). However, this method sets the same deadline for every query and is directly related to the throughput, so we have chosen to only examine throughput. Each experiment has been performed using 5, 10, 20, 50 and 100 parallel YCSB clients.

5.2 Experimental Results

In this section, the performance with and without the scheduler is compared in each graph; the dashed line is the performance of the baseline Cassandra and the solid line is the performance using our scheduler. The Cassandra cluster was restarted between every run. The results are averaged over 3 runs each.

A problem that arose somewhat often was issues of test environment consistency between experimental runs. That is, it is not entirely uncommon for a node to crash during operation. Initially, we did not examine the node status after each run, and therefore our initial results from the experiments could be quite skewed in favor of one method or the other. In addition, one of the nodes in the cluster did not have its time synchronized with the other nodes, resulting the schema disagreement issues that caused that particular node to drastically decrease its performance. After recognizing that this was causing skewed results we re-ran all of our experiments and took these issues into account.

The results of the 10GB experiment are shown in Figure 1. As can be seen, the version of Cassandra using the resource based query assignment did not perform nearly as well as the baseline version. However, the difference in throughput between the scheduled and baseline versions tended to stay constant throughout the course of each experiment. As well, the ratio of the difference between experiments was very similar. This seems to indicate that the overhead of performing the additional assignment logic is what resulted in the decrease in performance. We also monitored the servers resource usage during the course of the experiment and found that the server was never heavily loaded, which means that the performance difference is very minor (i.e., there is no noticable difference between 10% CPU usage and 12% CPU usage).

The results of running the 100% read workload on a 150GB database are shown in Figure 2. As before, the scheduled version of Cassandra is slower than the baseline version in every case. The difference is performance is approximately 100 operations/second the 5, 10, 20 and 50 client experiments; the difference rises to over 150 operations/second in the experiment that uses 100 parallel clients.

We believe that the poor performance of the scheduling al-

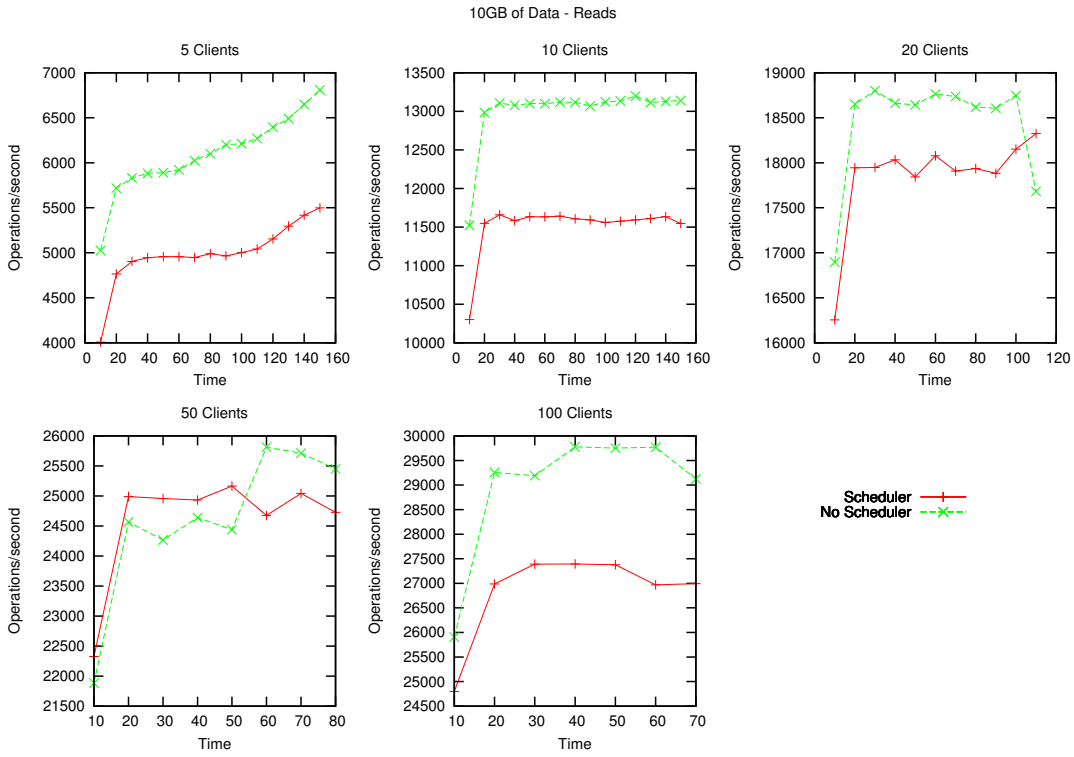

Figure 1: The experimental results of performing the 100% read workload on the 10GB database.

gorithm during the read workload is due to three reasons:

1. Even while 100 parallel clients were sending queries to the database, the node servers CPU usage and memory usage was very low. Therefore, the difference in throughput between the 150GB and 10GB databases (30000 operations/second versus 1900 operations/second) seems to indicate that hard-disk accesses are the bottleneck (this makes sense considering this is the bottleneck in traditional databases as well). This means that our resource usage formula is missing an important part of what determines how quickly a query can execute.

2. The introduced overhead of the scheduling logic seems to have a significant impact on our performance. This is especially apparent in the experiments on the 10GB database, where much of the data could be stored in the database cache. We believe that that this is what causes the throughput in the 10GB database experiments to differ by almost 1000 operations per second.

   Unfortunately, this is not something that can be easily resolved. The baseline version of Cassandra does not perform any kind of analysis of the replica servers and instead just sends the query to the first replica. In our algorithm we are adding an additional thread and network calls to Cassandra, as well as scanning a vector of length n (where n = number of replicas) and finding the replica with the smallest resource value. In isolation these actions do not cause a noticeable drop in performance, but when performing 30000 reads per second there is significant overhead.

3. Another reason why the scheduled read could never have better performance than the unscheduled version is because of the use of read digests [4] in Cassandra. A read digest is used to maintain consistency of replicas in the database. For every read query, Cassandra will read the value from the first replica, but also get a MD5 hash of the values on all of the other replicas. This is what Cassandra uses to gain eventual consistency. Naturally, a problem arises because even though we are reading the actual value from the least loaded replica, we are still querying the heavily loaded replicas to get a hash of the value.

   It is for this reason that we have also examined how our assignment operation performs when applied to the scan operation (which does not perform a read digest).

The results of the scans experiment on the 10GB database are shown in Figure 3. In these experiments the performance of both versions of Cassandra are very close to one another. The results are particularly close for the experiments with 50 and 100 concurrent clients. We believe this is because when

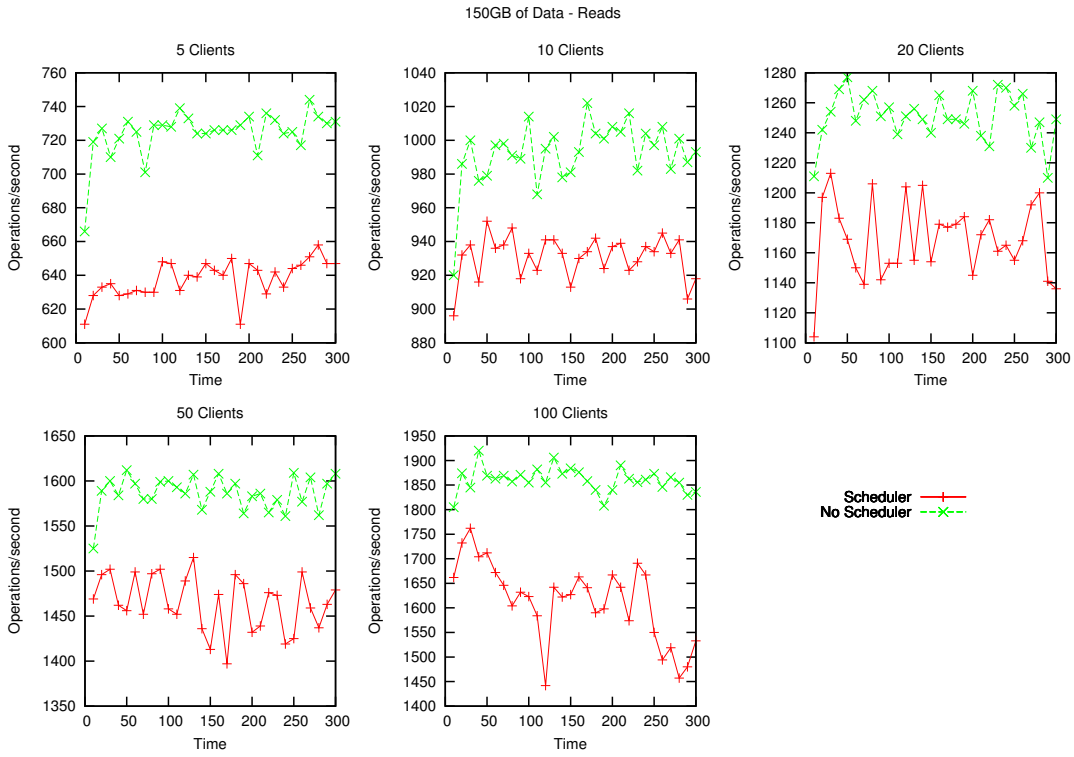

Figure 2: The experimental results of performing the 100% read workload on the 150GB database.

there are a high number of concurrent clients the CPU and memory usage varies enough between node servers that the query execution time is affected. We attempted to test with a greater number of clients than 100 (both 500 and 1000) and in both cases the results were similar because it appears that the master server became too heavily loaded.

Averaging the results of the experiments over 3 runs is not enough to say conclusively if the scheduled version of Cassandra provides better performance than the baseline version when performing scans with large numbers of concurrent clients. However, it appears that the performance of Cassandra when using the scheduling algorithm is at least on par with the performance of the baseline version of Cassandra. When the resource usage formula is improved to take factors other than CPU usage and memory usage into account it should, hopefully, provide consistently better performance than the baseline version of Cassandra. Confirming this requires further testing.

We also ran the 100% scans workload on the 10GB database while introducing artificial CPU and memory work into the server (this was done using the stress* tool). Every second node in the cluster was loaded by starting 30 processes that would continually malloc() and free() data. This resulted in fairly constant 100% CPU and memory usage.

*http://weather.ou.edu/ apw/projects/stress/

The results of the 5 and 10 client experiments are shown in Figure 4. In these cases you can see that the resource aware assignment has much better performance than the baseline Cassandra. This is because the resource aware assignment will consider that every second server is overloaded and not send queries to those servers. The results for the baseline version of Cassandra are taken from a single run of the YCSB. This is because the consequent runs all resulted in the YCSB throwing numerous exceptions before aborting. The figures for the cases with 20, 50 and 100 clients are not shown because performing experiments on the baseline version of Cassandra resulted in the YCSB benchmark continually aborting during the middle of the experiments. We believe this occurs because the queries sent to the heavily loaded server may be significantly delayed or lost, resulting in the YCSB believing the something is wrong with Cassandra (though we have not confirmed if this is actually the reason).

Although server load similar to the artificial load that we introduced may be quite rare in practice, we believe that this is the ideal environment to use our query assignment algorithm.

6. DISCUSSION

As can be seen, our approach only provides a noticeable improvement to the performance of Cassandra when multiple

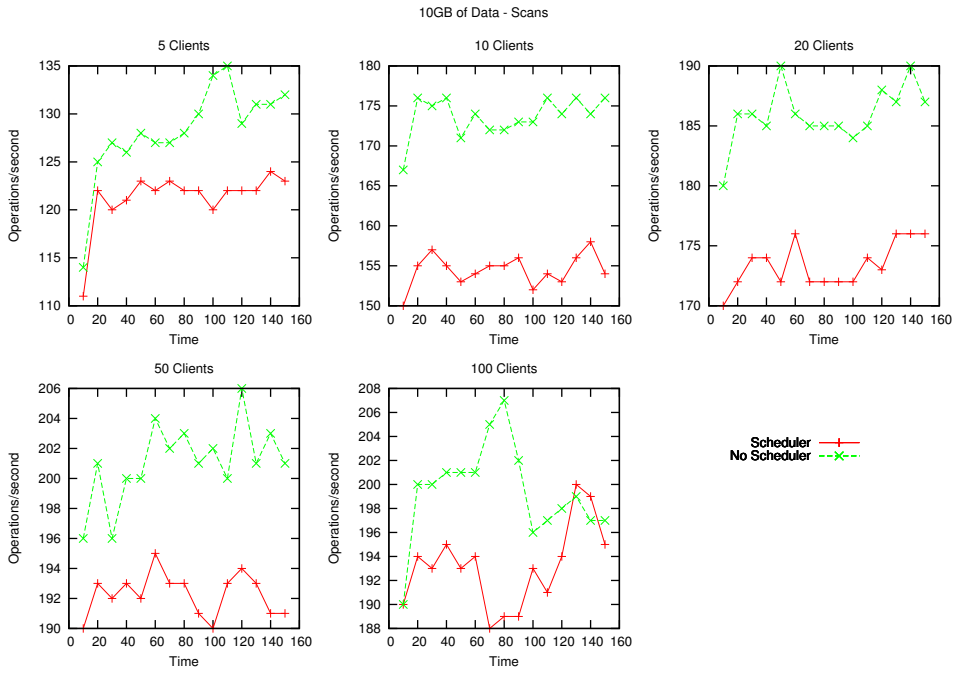

Figure 3: The experimental results of performing the 100% scan workload on the 10GB database.

nodes in the cluster are heavily loaded. This is not what we initially imagined our results would be. One of our intuitions is that the performance decrease exists because the resource usage formula does not accurately reflect what the time to execute a query will be. To confirm this intuition we performed an experiment that compared the resource usage to the query execution time in a Cassandra instance.

In the remainder of this section we will provide what we feel will be a more accurate resource usage formula for distributed databases, as well as discuss several limitations that may also have affected our work.

6.1 Re-examining the Resource Usage formula

To examine how representative the resource usage score is we have performed an experiment that compares resource usage to query execution time. The setup for the resource usage experiments is different than the setup for the primary experiments. This experiment was performed on a single Cassandra instance on a single machine with a 4-core 2.6Ghz processor and 8GB of memory. The server also contained a solid-state drive instead of the hard disk drive used in the servers in the cluster. The settings for YCSB and Cassandra were identical, except that the replication factor was set to *one*.

The CPU usage and memory usage was recorded every 250ms, and the query execution time was recorded for each query. The queries that executed in that 250ms window had their execution time averaged. The CPU usage and memory usage were used to calculate the resource usage of the server for each 250ms interval. The results were gathered over a 300 second interval using 10 parallel YCSB clients. Given our previous results showing that even 1000 parallel clients did not heavily load the node servers we do not believe that the results would be any different for a number of parallel clients greater than 10.

The results from the experiment are shown in Figure 5. There appears to be little correlation between the resource usage score and the query execution time. When the resource usage score is higher, there are slightly more outliers that require a longer execution time, but in most cases the execution time seems to be independent of the resource usage. In addition, the resource usage scores tend to either be very low or very high. While this experiment was running, we also periodically loaded the server with other work. This resulted in a higher resource usage score, but not in a higher query execution time.

This experiment seems to indicate that the current formula is not ideal for determining the resource usage of the server. However, we believe that this could still work if the formula was changed to reflect how some variables affect the query execution time much more. Additional variables can also be added that affect query execution time. For example, one of the primary bottlenecks in any database system is the hard disk, and we do not consider any variables related to the disk (e.g., disk access throughput).

Something to note is that the heap memory of the Cassandra instance rarely exceeded 1GB. Meaning that the memory

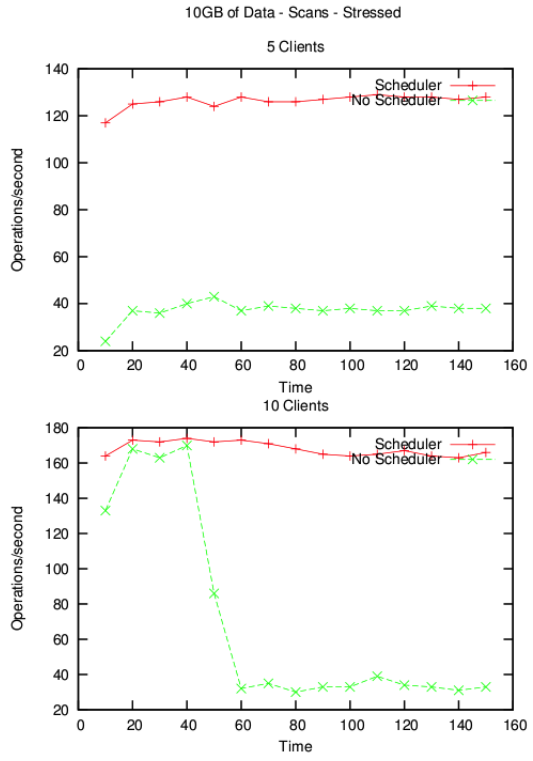

Figure 4: The experimental results of performing the 100% scan workload on the 10GB database (where every second node is stressed).

usage score had little effect during the normal experiments. Even when the server was being artificially loaded the heap memory therefore has little effect. This also means that little data is being cached, which increases the importance of measuring the disk load.

6.2 The New Resource Usage Formula

As Section 6.1 shows, the resource usage score is far from ideal. We have some ideas on how to create a better resource usage formula. For reference, the current resource usage formula is:

$$ResourceUsage = \frac{1}{1-CPU} \times \frac{1}{1-Memory}$$

The new resource usage formula is:

$$ResourceUsage = \frac{1}{1-CPU} \times j\left(\frac{1}{1-Memory}\right) \times k\left(\frac{1}{1-Disk}\right) \times \frac{1}{1-Distance}$$

6.3 Limitations

6.4 Future Work

7. CONCLUSION

Because the code that handles scans is completely disjoint from the code that handles reads, writes or updates it is plausible to implement our algorithm purely for scans while losing very little performance (just the overhead of a single thread in Cassandra that queries the CJD for resource information).

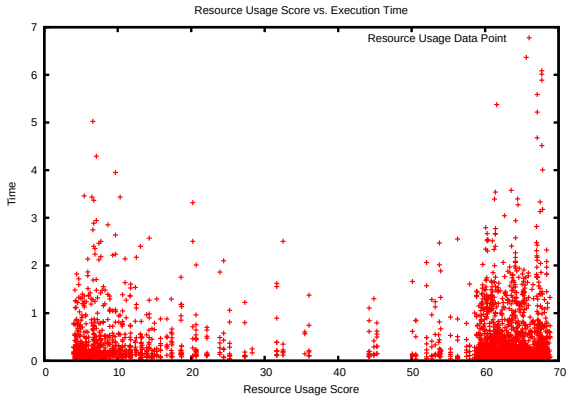

Figure 5: The experimental results comparing the resource usage score to the query execution time.

8. REFERENCES

[1] D. J. Abadi. Data Management in the Cloud: Limitations and Opportunities. *IEEE Data Eng. Bull.*, 32(1):3–12, 2009.

[2] R. Agrawal, A. Ailamaki, P. A. Bernstein, E. A. Brewer, M. J. Carey, S. Chaudhuri, A. Doan, D. Florescu, M. J. Franklin, H. Garcia-Molina, J. Gehrke, L. Gruenwald, L. M. Haas, A. Y. Halevy, J. M. Hellerstein, Y. E. Ioannidis, H. F. Korth, D. Kossmann, S. Madden, R. Magoulas, B. C. Ooi, T. O'Reilly, R. Ramakrishnan, S. Sarawagi, M. Stonebraker, A. S. Szalay, and G. Weikum. The claremont report on database research. *SIGMOD Rec.*, 37(3):9–19, Sept. 2008.

[3] D. Borthakur. *The Hadoop Distributed File System: Architecture and Design*. The Apache Software Foundation, 2007.

[4] A. Cassandra. Digestqueries. http://wiki.apache.org/cassandra/DigestQueries.

[5] B. F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears. Benchmarking cloud serving systems with ycsb. In *Proceedings of the 1st ACM symposium on Cloud computing*, SoCC '10, pages 143–154, New York, NY, USA, 2010. ACM.

[6] C. Curino, E. Jones, R. A. Popa, N. Malviya, E. Wu, S. Madden, H. Balakrishnan, and N. Zeldovich. Relational Cloud: A Database Service for the Cloud. In *5th Biennial Conference on Innovative Data Systems Research*, Asilomar, CA, January 2011.

[7] J. Ellis. Apache cassandra. Slideshare, 2011.

[8] A. Ganapathi, Y. Chen, A. Fox, R. Katz, and D. Patterson. Statistics-driven workload modeling for the cloud. In *Data Engineering Workshops (ICDEW), 2010 IEEE 26th International Conference on*, pages 87–92, march 2010.

[9] J. Gray, P. Sundaresan, S. Englert, K. Baclawski, and

P. J. Weinberger. Quickly generating billion-record synthetic databases. *SIGMOD Rec.*, 23(2):243–252, May 1994.

[10] A. Lakshman and P. Malik. Cassandra: a decentralized structured storage system. *SIGOPS Oper. Syst. Rev.*, 44(2):35–40, Apr. 2010.

[11] M. Meyer. The simple magic of consistent hashing. http://www.paperplanes.de/2011/12/9/the-magic-of-consistent-hashing.html.

[12] N. W. Paton, M. A. T. Aragão, K. Lee, A. A. A. Fernandes, and R. Sakellariou. Optimizing Utility in Cloud Computing through Autonomic Workload Execution. *IEEE Data Eng. Bull.*, 32(1):51–58, 2009.

[13] S. Z. Y. J. E. C. C. Wu, Eugene; Madden. Relational cloud: The case for a database service, Mar 2010.