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

Place abstract here

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 [3]. 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 [2]. However, the modern scheduling systems only simply assigned query to the closest replica [4], 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 [10, 8, 6, 1]. 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 ïňĄnite 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) [9] 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 [?]). 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 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 are different kinds of workloads to query the data in 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 [?] 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

TIn 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 b and tetween every run. The results are averaged over 3 runs each.

The results of the 10GB experiment are shown in Figure X 1. As can be seen, the version of Cassandra using the resource based query assignment did n 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).ot perform as well as the baseline Cassandra in any of the five experiments.

The results of the 150GB experiment are shown in Figure Y 2. Even with 100 parallel clients sending queries to the database, the node servers CPU usage and memory usage was still quite low. It seems that hard-disk accesses are the major bottleneck, which we have not accounted for in our method. The Cassandra read-digest [?] also accounts for the slow performance, as even though you may read the data from the lowest resource usage server, you still need to access the other two servers in order to get the hash value and timestamp for the data.

The introduced overhead of managing the hash table also seems to be at least partially responsible for the poor performance. In the baseline version, Cassandra does try to access the list of endpoints, and only sends the query to the first endpoint. In our version, we are not only traversing the list of valid replicas, but also swapping entries in that list (to sort it by resource usage). We are also storing the resource usage values in a Java HashMap, which starts to perform poorly when many threads are accessing it at one time (this is apparent from the extremely poor performance of the 100 client experiment).

Because of the read digest, you still need to go to the poor performing servers to get a hash value. Running the experiment using scans shows that this is not always the case.

We only ran the scan workload with 10GB of data. As the performance of the scheduler when performing scans was no better than the performance without then it is clear that the scans do not work. The primary difference between scans

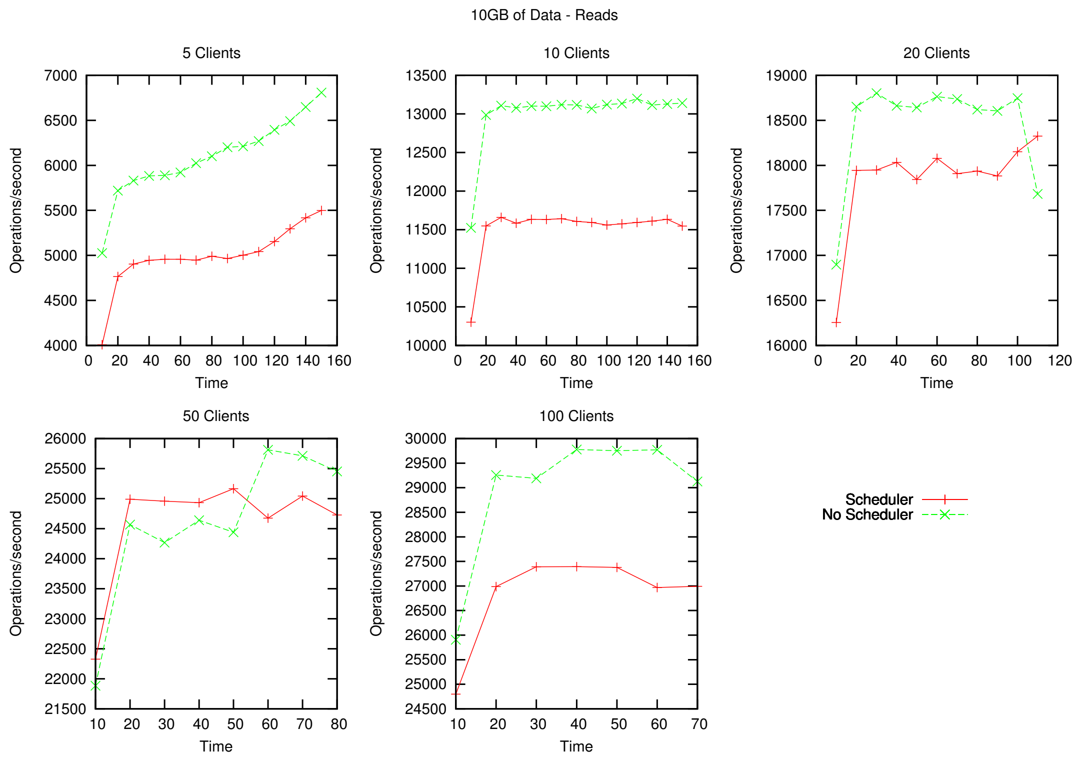

Figure 1: The experimental results of performing the 100% read workload on the 10GB database.

and reads is that the scan does not perform a read digest, thus removing the need to call every server to retrieve the value for a single key. The results of the 10GB experiment are shown in Figure Z.

We also ran the scans workload while introducing artificial workload into the server. This is a fairly contrived case, but is the ideal case for our work. The results of the 5 and 10 client experiments are shown in Figure A. 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. The figures for the cases with 20, 50 and 100 clients are not shown because performing experiment on the baseline version of Cassandra resulted in the Cassandra code continually crashing.

6. DISCUSSION

[TODO: Need to fill in this with a small intro sentence]

6.1 Re-examining the Resource Usage formula

As can be seen, our approach yields a small improvement to the baseline Cassandra performance. However, this improvement is not nearly as good as we had hoped. Our intuition is that the performance is gains are minimal because the formula that we have used to calculate the resource usage score does not totally suit our purpose [?]. To confirm this intuition we performed an experiment that compared the resource usage to the query execution time in a Cassandra instance.

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. We do not believe that the results would be significantly different for other numbers of parallel YCSB clients, but we have not confirmed this.

The results from the experiment are shown in Figure A. There appears to be little correlation between the resource usage score and the query execution time. When the re-

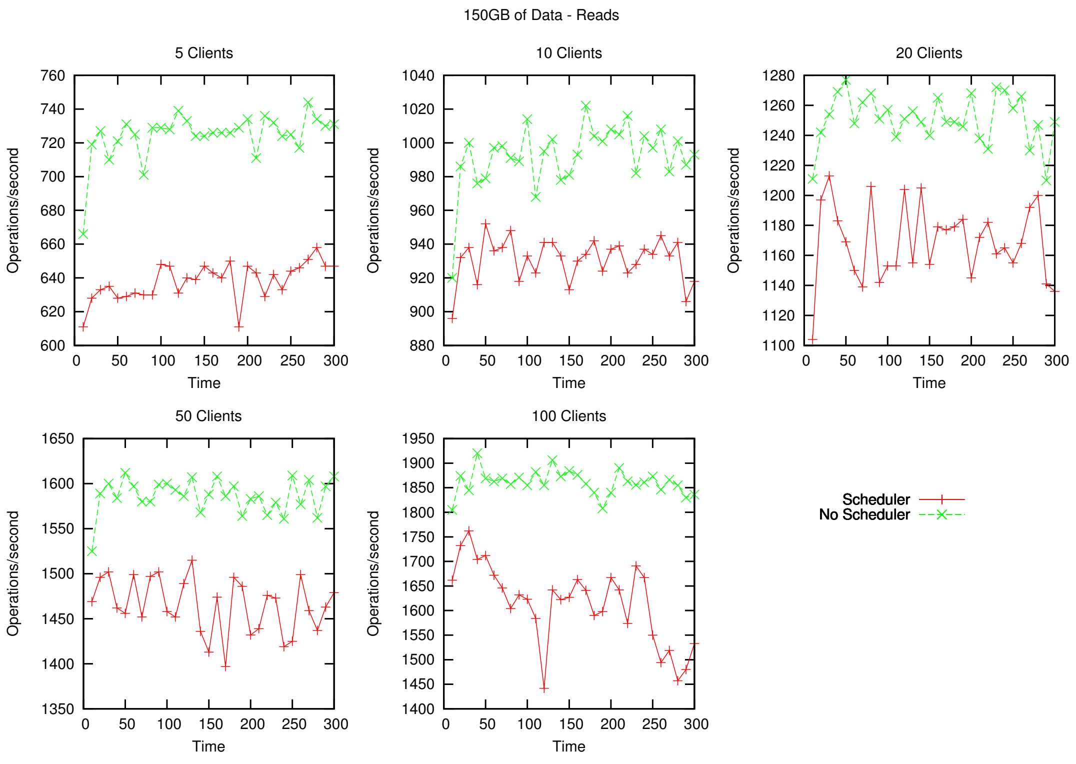

Figure 2: The experimental results of performing the 100% read workload on the 150GB database.

source 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 were also attempting to artificially load the server. However, this seemed to make little difference to the resource usage score.

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 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.

7. FUTURE WORK

8. CONCLUSION

9. REFERENCES

[1]

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

[3] 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.

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

[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.

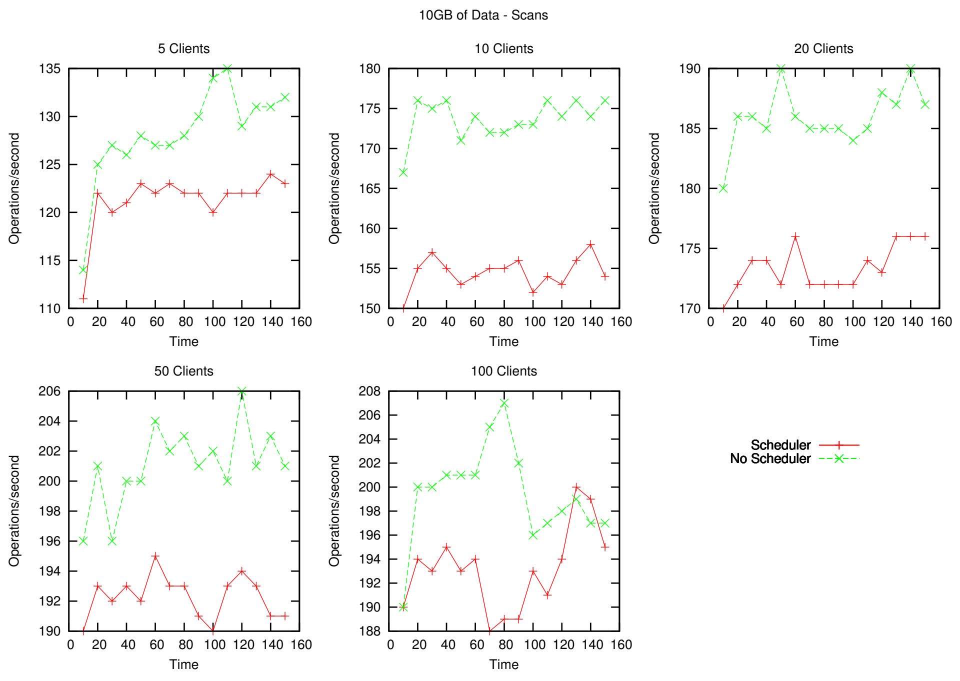

Figure 3: The experimental results of performing the 100% scan workload on the 10GB database.

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

[10] 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.

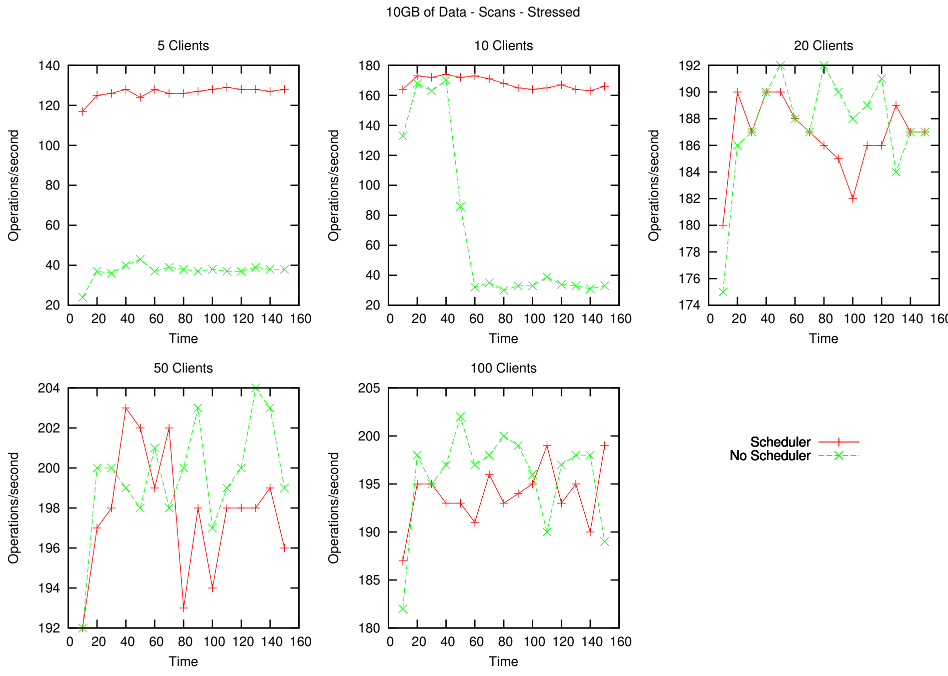

Figure 4: The experimental results of performing the 100% scan workload on the 10GB database (where every second node is stressed).