Cassandra on Blue Waters

Yosub Shin

University of Illinois at Urbana-Champaign

Abstract

Distributed key-value stores running on commodity machines have become very popular, as cloud computing has become affordable and approachable. However, possibility of running such systems in high performance machines has been overlooked. We investigate the characteristics of a distributed key-value store running on a supercomputer and compare with the distributed key-value store running on commodity machines.

1. Introduction

Characteristics of distributed key-value stores running on commodity machines have been investigated thoroughly by various benchmarks such as [18] and [26]. However, as far as we know, no work has been done on evaluating performance and investigating possible bottlenecks on running a distributed key-value store on a supercomputer. Since supercomputers have very different characteristics in terms of computing power, memory, network bandwidth, and disk I/O than those of commodity machines, we expect to see different characteristics if we run a distributed key-value store on a supercomputer.

Therefore, we investigated the characteristics of a distributed key-value store running on a supercomputer. Specifically, we ran Apache Cassandra, a popular distributed key-value store, on Blue Waters [4] supercomputer. As a comparison, we also ran Cassandra on Emulab [8] cluster, which provides variety of commodity machines on private cloud.

We initially ran Yahoo! Cloud Service Benchmark (YCSB) [18] to compare latency-throughput characteristics of the workloads under different kinds of machines. We found that as the throughput approaches the saturation point, the latency rapidly increases because of the underlying bottleneck of the Cassandra system. Then, we measured the throughput and average latency of Cassandra on Blue Waters and Emulab as we increased the number of machines. We verified that the Cassandra scales linearly for both Blue Waters and Emulab. We also showed that the throughput of the Cassandra system on one Blue Waters compute node is equivalent to that of seven Emulab machines, implying that Blue Waters is more cost-effective in terms of the throughput of Cassandra. In the same experiment, we showed the latency of Cassandra scales well on both Blue Waters and Emulab. Next, we compared the characteristics of read and write latencies of requests in Cassandra on both Blue Waters and Emulab cluster. We observed that the average read and write latencies of Blue Waters are lower than that of Emulab, which can be described by the supercomputer's superior interconnect. However, the Blue Waters showed higher tail latencies for both read and write queries. We also utilized the Probabilistically Bounded Staleness framework [15] to evaluate the consistency of Cassandra in both environments. In a fair setup where same number of cores were used to store equivalent amount of entries in both environments, we observed that the consistency of the Emulab is higher, which can be explained by the relatively long latency tails in Blue Waters.

*shin14@illinois.edu

The structure of this paper is composed of following. In section 2, we provide brief summaries on topics covered by this paper, such as distributed key-value store, Cassandra, supercomputer architecture, and YCSB. After that, in section 3, we discuss the motivations of this research in detail, including the challenges we are trying to solve. In section 4, we demonstrate the experimental results along with the methods used in our experiments. Next in section 5, we discuss our results in the context of the possible future work. Finally, in section 6, we summarize our findings from this research and discuss its implications.

2. Background

2.1. Distributed Key-Value Stores

Distributed key-value stores have become very popular in last 10 years. Large companies rely on key-value stores to store customer transactions, user profiles, and other business critical data. With the fast growth of internet users and geo-distributed nature of internet services, more companies are moving toward adopting distributed key-value stores which can be operated in multiple geo-distributed data centers.

Systems like Cassandra [23], BigTable [17], HBase [12], Dynamo [19], and MongoDB [14] are just a few examples of well known distributed key-value stores. In accordance with CAP theorem [21], which proves that not all three of consistency, availability, and partition tolerance can satisfied at the same time, each distributed key-value store promises different guarantees and is suitable for certain situations. BigTable and HBase offer high consistency at the price of availability under network partition, whereas Cassandra and Dynamo promises availability and partition tolerance at cost of consistency. Different key-value stores are also optimized for different workloads. Cassandra and HBase adopt log-structured data structure to optimize write performance significantly by utilizing sequential writes on disks. However, this comes at cost of read latency because writes are fragmented into multiple pieces.

2.2. Cassandra

Among various distributed key-value stores, we intend to use Cassandra for our research. Cassandra is a decentralized key-value store with high availability and partition tolerance, and boasts high write throughput because of its utilization of log-structured merge tree [25] as its data structure. Unlike HBase, Cassandra does not have a notion of master node.

When a client requests an entry, any machine in the cluster can serve as a coordinator. A coordinator in Cassandra is a physical machine that receives requests from a client and contact other machines in order to serve the client's request. The data placement in Cassandra utilizes consistent hashing as in the Chord system [29]. The consistent hashing algorithm uses a virtual hash ring, which wraps around at the end of the hash range. Each machine in Cassandra owns one or more tokens uniformly distributed on virtual ring positions. When a key is being looked up in the hash ring, a partitioner iterates through the ring in the clockwise direction until it encounters a token. The machine that owns this token is responsible for the entries of the key, and therefore contacted by the coordinator. The contacted machine searches for the given key in its memory or local file system, and returns its value or notify to coordinator that it does not have an entry for the key. At the coordinator, it waits until enough number of replicas respond to the request, and compares their responded values. The number of replicas for which coordinator waits is determined by tunable consistency level. Once the consistency level is satisfied, the coordinator finally returns the requested entry to the client, and the query is over.

As briefly mentioned earlier, Cassandra internally uses a log-structured merge tree called SSTable, to persist data. When a write request arrives at a coordinator, the request is routed to an appropriate machine similar to the read path explained above. Once a machine receives a write request from the coordinator, it is initially stored in a data structure called Memtable, and also appended to commit log for durability. Memtable is an in-memory data structure that temporarily store en-

tries in sorted manner for fast access to recent entries. When the size of the Memtable exceeds a threshold, it flushes itself into a new SSTable. SSTable is an immutable sorted data structure designed for efficient write performance into a hard drive. Since it is immutable and a Memtable is already sorted, flushing SSTable can be done only using sequential writes. Utilizing SSTable is very beneficial for write-heavy workloads since no random disk I/O is incurred, as shown in [18] in comparison to traditional RDBMS. However, this benefit comes at the cost of worse read performance. If many partial updates are executed on a single row, and if a client wants to read the entire row, Cassandra has to aggregate many pieces of the entry that are fragmented in multiple SSTables in order to serve that entry to the client. This naturally degrades the read performance, and it is also demonstrated in [18].

In order to overcome read performance degradation, a technique called compaction is used. The compaction is first introduced by [17], and it is being widely used in almost every log-structured database system. It is an operation that merges multiple smaller SSTables into one large SSTable such that fragmented entries are aggregated and deleted entries are removed to save space. Although compaction does not requires random I/O operations, it still incurs a large overhead to the system. Due to its overhead, there are multiple options for compaction algorithms in Cassandra. Size tiered compaction algorithm inspired by [17] compacts similarly sized SSTables and saves I/O and suitable for write-heavy workloads. Leveled compaction, inspired by [13], incurs more disk I/O, but good for workloads with many fragmented updates.

2.3. Supercomputer Architecture

A typical supercomputer is consist of compute nodes, interconnects, and a storage system. A compute node is a basic building block that does the computational processing, which is consist of CPU, memory, and network cards. Each compute node is built into a physical blade.

Another major component of a supercomputer is the interconnects. Unlike commodity clusters

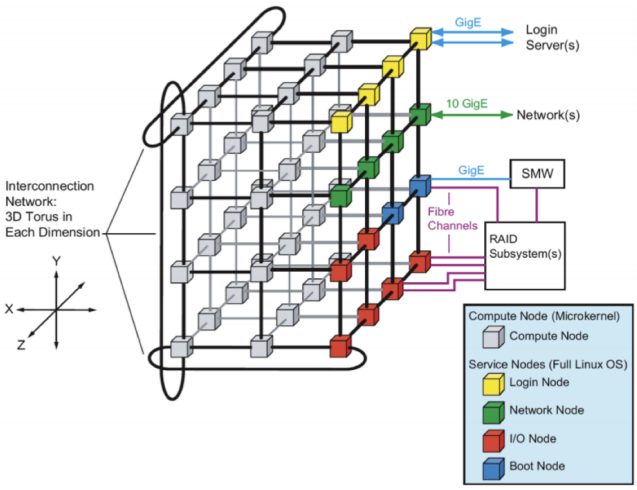


Figure 1: Cray XE6 supercomputer architecture[30]. Each compute node is connected with 6 neighboring nodes in x, y, and z directions, forming 3D Torus topology. In Blue Waters, this topology has dimension of $24 \times 24 \times 24$, making it one of the largest supercomputers in the world.

where a single or a few central switches connect multiple servers, a supercomputer connects each compute node to its neighboring compute nodes in a very tight physical space in order to minimize latency and provide homogeneity of communication. Therefore, careful physical layout of the interconnect between compute nodes is crucial to the design of a supercomputer. For example, Blue Waters supercomputer uses 3D Torus structure as shown in Fig. 1, in which each compute node is connected with six neighboring nodes along x, y, and z axes. In Blue Waters, the 3D Torus is consist of logical cube of dimensions $24 \times 24 \times 24$, and each interconnect has the bandwidth of 9.6GB/s [3].

In typical supercomputer systems, storage is physically separated from compute nodes. This is to ensure tight interconnect and sufficient cooling of compute nodes. Each storage request originating from a compute node arrives at an available I/O node. An I/O node is also located in the 3D Torus to provide fast access from compute nodes. It is responsible for routing the request to the storage system through fiber channels.

2.4. Key-Value Stores on Supercomputers

There are some previous works done by other researchers to benchmark or implement distributed

key-value stores on supercomputers. [22] has benchmarked Accumulo [1], an open source implementation of BigTable, on MIT SuperCloud [27]. They demonstrated the peak performance of 100,000,000 database inserts per seconds by running Accumulo on TX-Green supercomputer. However, they did not compare with the performance of Accumulo running on a baseline cluster, therefore unable to verify a bottleneck of a distributed key-value store running on supercomputer.

NoVoHT [16] introduces a new key-value store specifically designed for supercomputers. It claims significant performance gain over traditional key-value stores such as LevelDB, but its main focus regarding supercomputers seems to be on less dependency on Linux libraries which are normally missing in supercomputer operating systems.

2.5. YCSB

Yahoo! Cloud Service Benchmark [18] is one of the most widely used benchmark tools for evaluating distributed key-value storage systems. YCSB is consist of load phase and execute phase. In load phase, YCSB injects synthetic key-value pairs into the designated key-value store. The user can specify which database binding to use, how many rows to insert, number of columns in a row, and size of each column to simulate the desired workload.

In execute phase, YCSB issues requests according to the predetermined distribution(zipfian, uniform, or latest), read-write ratio, desired operations rate, number of client threads, and number of operations. It is important to note that YCSB adopts the closed loop request model [28]. That is, each YCSB client thread waits until the request it issued responds back, before issuing another request. Because of this, it is necessary to spawn multiple YCSB client threads to achieve high operations rate. For example, if we want to evaluate a key-value store at 300,000 ops/s operations rate, and if average request latency is 5ms, we need at least $300000 \times 5/1000 = 1500$ threads concurrently running YCSB workload.

3. Motivation

Traditionally, supercomputers have been used in intensive scientific computing tasks. As of this writing, the most significant tasks in computational resources being executed on Blue Waters are Biophysics(25.2%), Stellar Astronomy and Astrophysics(15.8%), and Climate Dynamics(13.4%) [4]. One of the characteristics of typical scientific computing tasks is that they are batch jobs that take a lot of CPU cycles. However, since batch tasks are scheduled on a job queue and each task takes a few hours or even a few weeks, getting a result immediately in order of milliseconds is not normally desired nor possible.

Therefore, running a latency sensitive key-value store may seem to be a counter-intuitive idea. However, we claim that the fundamental reason why most supercomputing jobs are batch tasks and not interactive is merely a cost efficiency reason. Most small companies cannot afford to build or own a supercomputer that costs 200 million dollars. However, if we turn our attention to giant internet companies such as Google, Facebook, and Amazon, it is well known that these companies build their own private data centers [10, 9]. These are companies that have millions of users querying their websites every minute. If the supercomputer architecture can provide the performance that these companies require at a reasonable cost, they might choose to build a new data center that houses a powerful supercomputer instead of many commodity servers. Furthermore, if one can achieve 10x or better performance by adopting supercomputer architecture, it could possibly create the entirely new use cases that were previously unthought of.

3.1. Achieving High Throughput and Low Latency

One of the most important goals of the distributed key-value storage systems is to achieve high operations rate and low latency at the same time. However, it is known that there is a trade-off between these two metrics [18]. That is, as a machine gets more overloaded with higher throughput, it exhibits higher latency, and the rate of latency increase is very high such that throughput does not increase past certain point. If this happens, the quality of service to the user is degraded and this is undesirable in the production environments.

Therefore, in order to achieve better throughput, there are two typical approaches we can take. One is to replace the existing machines with beefier machines with higher computation power, which is usually called a scale-up approach. Another approach called scale-out is to deploy more of the less powerful machines in parallel, such that the aggregated operations rate of the whole system increases. There are many studies that investigate this issue [31],[24], but the general wisdom is to first scale up as much as possible. When the amount of data and throughput requirement is too great, then we have no other choice but to scale-out to achieve better throughput level. Our experimental results also back this general wisdom as single powerful Blue Waters machine outperforms many Emulab machines deployed in parallel.

3.2. Cost Effectiveness

Even though scale-up is a preferred approach over scale-out, the reality is that many companies take scale-out approach because of the cost effectiveness. As the machine gets more powerful, the price of the machine increases super-linearly. This was also studied in [31] and the authors claim that in some cases, scale-up machine is more cost effective because of the gain of performance is too great. We also conduct similar analysis of the cost efficiency of running Cassandra on supercomputer and on commodity cluster. The cost of Blue Waters supercomputer is $188 million [5], and it houses 26864 compute nodes[3]. If we simply divide the total cost by number of compute nodes, we get unit cost of $6998/node. On the other hand, the cost of Dell PowerEdge R730 which is an successor model to the R710 model used in Emulab cluster costs $2159 [7]. Ignoring the cost of switches and other infrastructure required to build a commodity cluster, we can safely estimate that the cost of one Blue Waters compute node is not greater than the cost of four Emulab machines. Therefore, if we can demonstrate that the Cassandra system running on n Blue Waters compute nodes performs better than that of $4n$ Emulab machines, we can argue Blue Waters is more cost-effective than Emulab. In a latter section, we experimentally show that one Blue Waters node outperforms 7 Emulab machines, which implies that Blue Waters is more cost effective in some metrics.

3.3. Unique Architecture of Supercomputer

As mentioned previously, architecture of a supercomputer has a few very different characteristics from commodity cluster. First, a supercomputer has extremely high performance interconnects that connect among different compute nodes and between compute nodes and storage layer. Specifically, Blue Waters uses 3D Torus topology to connect neighboring compute nodes are interconnected along x, y, and z axes with each interconnect's bandwidth being 9.6GB/s [3]. This is very different from commodity cluster's network topology where a central switch routes inter-machine traffics. Having extremely fast and high bandwidth interconnect means that communication among Cassandra cluster is much faster and more scalable.

Second, a typical supercomputer does not have a local disk attached to each compute node. Instead, there is a separate storage layer which utilizes a parallel file system such as LustreFS. Use of parallel file system enables extremely high I/O throughput. In Blue Waters, aggregated I/O bandwidth is approximately 1TB/s [3].This poses a challenge to our attempt at deploying Cassandra in Blue Waters. Even though Blue Waters provide excellent I/O bandwidth, there are occasional spikes in I/O latencies as a parallel file system optimizes requests by processing them in batch, and access of a file should wait for multiple Acks from distinct storage devices. Since Cassandra assumes the locally attached storage, we occasionally ran into internal exceptions resulted from latency spikes. Therefore, in our experiments, we decided to use a ramdisk for our storage option for Blue Waters. For fair comparison, we also used the ramdisks on Emulab machines. Although using ramdisk for storage means that the data is lost upon a machine failure, the use of ramdisk can be justified, because some of the newer supercomputers such as Gordon [11] are equipped with SSDs that are locally attached to each com-

pute node. If we have a supercomputer with such architecture, we can replace ramdisks with locally attached SSDs and thus achieve fault tolerance.

4. Experiments

4.1. Hardware & Experimental Setups

For our experiments, we used Blue Waters supercomputer and Emulab cluster. For Blue Waters, we used XE compute nodes, each with 16 cores, and 64GB memory. The nodes are interconnected in 3D Torus topology as shown in Fig. 1. The operating system used is Cray's custom built Linux in Cluster Compatibility Mode(CCM) [6]. For Emulab cluster, we used d710 nodes, where each machine has 4 cores, 12GB memory, and 6 units of Gigabit NICs. It runs Ubuntu 12.04 operating system.

For the first two experiments in section 4.2 and 4.3, we used Cassandra version of 2.1.3, and the settings used were replication factor of 1, and read/write consistency level of 1. Additionally, we used SimplePartitioner for partitioner and SizeTieredCompactionStrategy for compaction strategy. As mentioned in the previous section, we used ramdisk as the storage option for both Blue Waters and Emulab. We used 1 to 17 Blue Waters and Emulab machines to run Cassandra for our experiments.

In our latency CDF and consistency experiments in section 4.4 and 4.5, we used Cassandra 1.2.8, and we used replication factor of 2, and read/write consistency level of 1. This is because the PBS predictor [15] we used only supports Cassandra version of up to 1.2.x. Other settings for Cassandra were identical with the first set of experiments.

For YCSB, we used the uniform workload type, 95:5 read and update ratio. The column family used for our test had size of each row of 1KB and there were 10 columns. We loaded 1GB of data in each Cassandra node which is equivalent to 1,000,000 entries per node. In order to avoid overloading Cassandra servers, we limited the number of connections to each core to 8 as it was done in [26]. For Blue Waters, the number of connections per compute node was set at 128, and it was set to 32 for Emulab server. In our test, as we increased the number of connections further, we observed that the throughput was unstable. The total number of YCSB client threads varied from 32 to 544 for Emulab, and 128 to 2176 for Blue Waters. The maximum number of YCSB client threads at each machine was set at 250 and 125 for Emulab and Blue Waters, respectively. This was due to the administrative reason that higher number of threads were prohibited by the Blue Waters system. The measurements were done for 60 seconds after a few seconds of warm up period, throughout the multiple YCSB worker machines in parallel. In our test, we observed that creating too many connections at the same time caused Cassandra servers to misbehave. Therefore we created connections by injecting a few seconds of delay per YCSB worker machine. Since we used multiple worker machines, we calculated the aggregated throughput by adding up throughput at each machine and normalizing it by number of operations injected on that machine. Aggregated average latency was calculated similarly by taking average of each latency normalized by number of operations.

4.2. Throughput-Latency Relations for BW and Emulab

In order to demonstrate the throughput and latency relationship, we measured average latency of read requests after fixing the target operations rate for each data point in Fig. 2. Then we gradually increased the target throughput until it reaches the saturation point. At throughput saturation point, the latency increases very rapidly past some throughput level. This means that the Cassandra server has reached a bottleneck in one of its request pipelines due to too many concurrent requests, such that each request is queued for longer time. We have not yet investigated the underlying cause of bottleneck, but we plan to do so in the future experiments. Some possible causes of the bottleneck are CPU resources and network I/O. In realistic situation where a hard drive or SSD would be used as the storage option, storage I/O may also be a bottleneck, but it is out of scope of our work, because of the constraints on

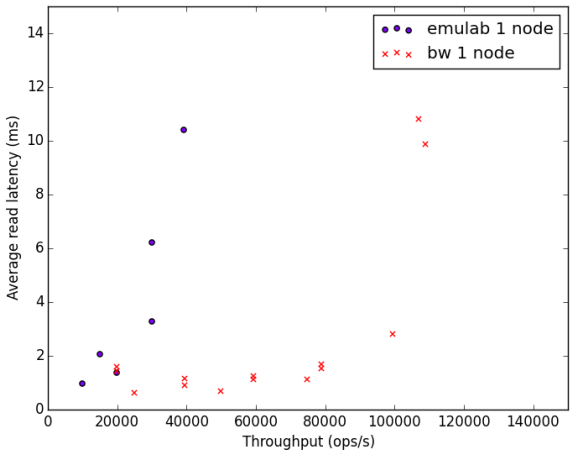


Figure 2: The relationship of throughput and average read latency of Cassandra on one Blue Waters compute node and one Emulab machine. As throughput approaches the saturation point, the latency increases rapidly, indicating a bottleneck in Cassandra. Saturation throughput value of Blue Waters is at least twice as much as that of Emulab.

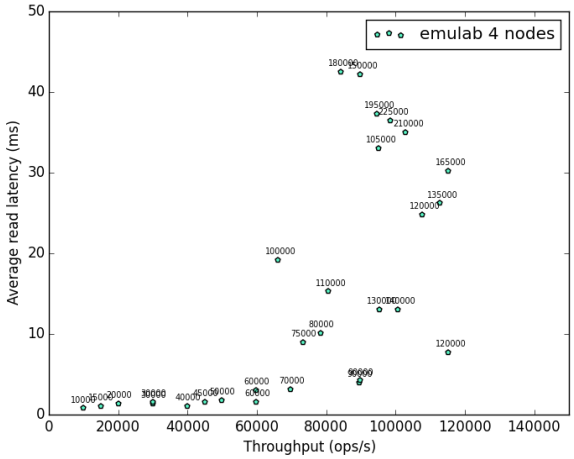


Figure 3: When number of connections per Cassandra server is not constrained, the latency and throughput are unstable. Text labels on data points correspond to the target throughput for that experiment. Some data points near the saturation throughput exhibit degraded throughput and latencies.

the current supercomputer architecture, as it was discussed in section 3.3.

Fig. 2 demonstrates the throughput and read latency relationship in one node Cassandra cluster for Blue Waters and Emulab. For both environments, average latency is below 1ms when throughput level is low. However, as throughput approaches at its saturation level, the latency increases very rapidly and therefore prevents throughput from further increasing. This result is consistent with that of [18]. We could observe that for one machine Cassandra cluster, the maximum throughput of Blue Waters node is much better than maximum throughput of Emulab machine by more than factor of 2. This is understandable as each compute node of Blue Waters has 4 times more cores and memory. It is important to note that the maximum throughput described here is not a stable value, but rather a fluctuating one.

In this experiment, we did not constrain the number of connections per Cassandra server, such that we can fully understand the latencies characteristics near throughput saturation points. However, we observed that if we do not limit the number of connections for Cassandra servers, we get very unstable throughput and latency results as in Fig. 3, especially with multiple Cassandra nodes. Therefore, in our other experiments, we controlled the maximum number of connections per Cassandra server in proportion to the number of cores in the machine.

4.3. Scalability

4.3.1. Throughput

In this experiment, in order to avoid overload of the Cassandra servers, we limited number of connections at each server at 128 and 32 at Blue Waters and Emulab, respectively. Several measurements in Fig. 3 from the previous experiment suggests that higher throughput can be achieved at the cost of higher latency and unstable throughput. However, we limited the number of connections because the purpose of this experiment is to demonstrate the linear scalability under increasing number of Cassandra servers.

In Fig. 4, we show overall throughput achieved given the number of Cassandra server nodes. We measured the overall throughput of Cassandra on Blue Waters and Emulab while increasing the number of machines. The number of Cassandra machines ranged from 1 to 17, and each data point was measured at least three times. The throughput of both environments suggest that they scale

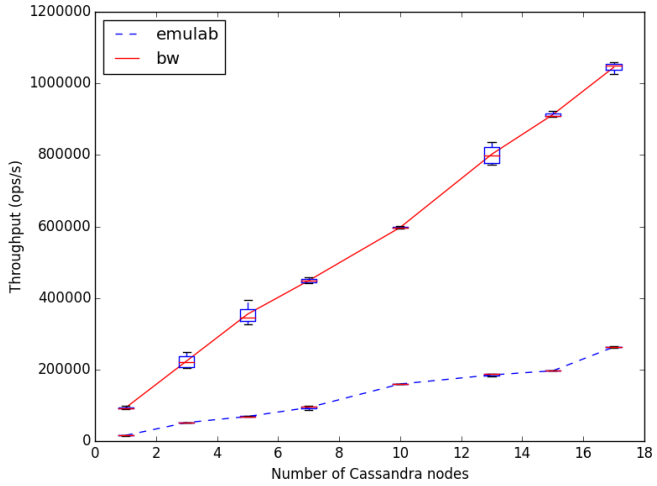


Figure 4: Overall throughput of Cassandra on Blue Waters and Emulab as number of Cassandra servers changes. Both Blue Waters and Emulab exhibit linearly scaling throughput. 7 Emulab machines are equivalent to 1 Blue Waters compute node in terms of overall throughput level.

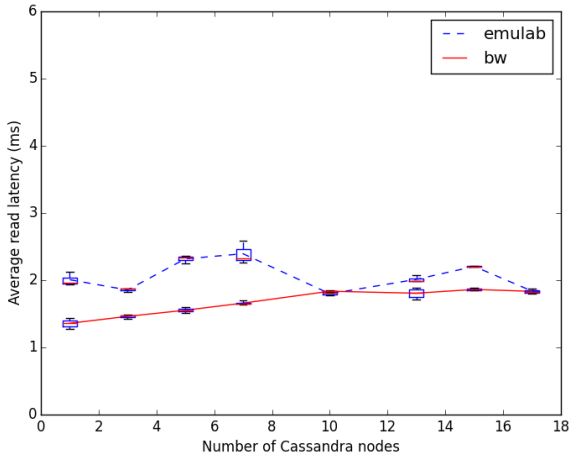


Figure 5: Latency of Cassandra on Blue Waters and Emulab as the number of Cassandra servers increases. The latencies are stable for both Blue Waters and Emulab, implying Cassandra is scalable in terms of latency.

linearly in the number of Cassandra nodes. Another important point to note is that the throughput of one Blue Waters machine is equivalent to the throughput of seven Emulab machines. Comparing this with the number of core ratio of 4:1, we conclude that Blue Waters has much higher per core performance in terms of throughput. Also, if we consider the cost ratio of each Blue Waters compute node and Emulab d710 server of 4 to 1, the throughput ratio of 7:1 is still greater, which implies that Blue Water is more cost effective in terms of stable throughput achieved.

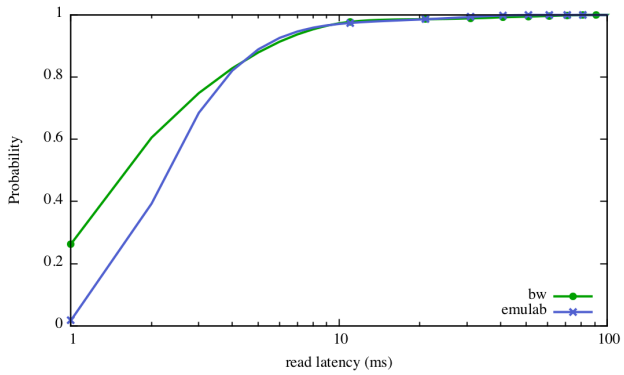
4.3.2. Average Latency

In this section, we measured average request latency under varying number of Cassandra nodes. We also controlled the number of connections at each server as the previous experiment. In Fig. 5, for both Blue Waters and Emulab, the average read latency is stable across the increasing number of Cassandra nodes. Also, Blue Waters generally performs at lower latency level despite the fact that absolute number of connections is four times greater and throughput is much higher at Blue Waters than in Emulab. We will explain the latency characteristics of Cassandra under Blue Waters and Emulab in more detail in section 4.4.

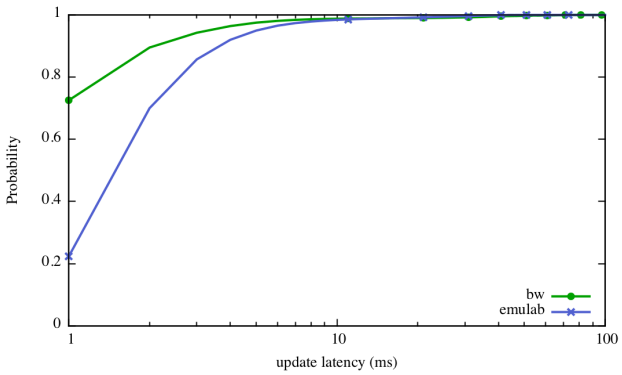
4.4. CDF of Latency

In this section, we compare the read and write latencies of Cassandra at different percentiles on Blue Waters and Emulab. For fair comparison, we determined the number of Cassandra machines for Blue Waters and Emulab such that total number of cores utilized in the cluster is same. In our experiment, we used 3 Blue Waters XE compute nodes and 12 Emulab d710 machines, since a single XE node has 16 cores and a single d710 node has 4 cores. Also, we set the overall target throughput to 60,000 ops/s which is below the saturation levels for both environments.

In Fig. 6a, we observe that the read latency at Blue Waters is lower than Emulab at lower percentiles from 0% to 80%. Above 80 percentile, both Blue Waters and Emulab share similar tail latencies. In Fig. 6a, we observe lower latencies for Blue Waters at across most latency percentile regions. This can be explained with two reasons. First, the physical distances between compute nodes in a supercomputer is much closer than commodity servers by its design, and since any network latency is bounded by speed of light, more densely laid out machines imply lower latency. Second, topology of network interconnect in a supercomputer is much denser than in a commodity cluster. In Blue Waters, each compute



(a) Read latency CDF



(b) Write latency CDF

Figure 6: The read and write latency CDF of Cassandra under Blue Waters and Emulab. For fair comparison, the same overall number of cores were utilized. X-axes are in logarithmic scale.

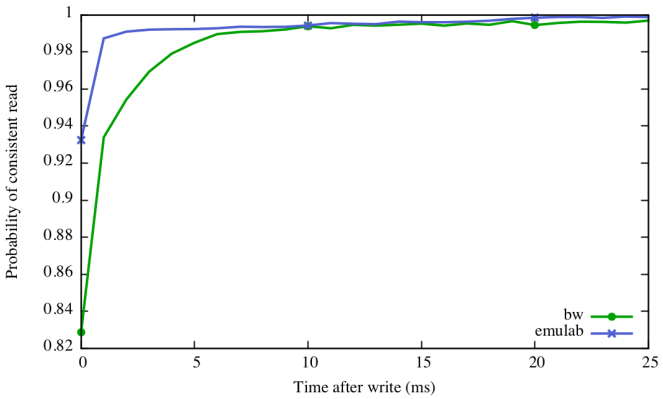


Figure 7: The probability that a query that was issued t ms after the write request on the same key will read the most recent version. (N,R,W) employed in this plot is (2,1,1).

node is connected to 6 neighboring compute nodes, and the nodes are forming highly interconnected cube as depicted in Fig. 1. This highly interconnected topology enables much lower latency thanks to less number of hops required for a packet, and also very high throughput because sheer number of interconnect is very large.

4.5. Consistency

In this section, we utilize the Probabilistically Bounded Staleness(PBS) framework [15] to evaluate how consistent Cassandra is in Blue Waters and Emulab. PBS is a framework that models the probability of the stale reads in a quorum based distributed system. The authors of [15] have instrumented Cassandra to estimate the probability of reading the most recent version at t times after the write request was issued by profiling the read and write requests. We employed this technique to compare the consistency of Cassandra in Blue Waters and Emulab. For this experiment, we set the replication factor to 2 (N=2), the read consistency level to 1 (R=1), and the write consistency level to 1 (W=1). In Fig. 7, we portray the probability of consistent read—the probability of reading the most recent version at t ms after the write was done—for Blue Waters and Emulab.

To our surprise, the Emulab cluster showed higher probability of consistent reads across the time t than Blue Waters. Combining this with the result in section 4.4, we deduce that the low consistency level achieved by Blue Waters is due to high variability of read latencies, which is suggested by its relatively long latency tail in read-latency-cdf.

5. Future Work

5.1. Large Scale Cluster

In our experiments, we have used at most 17 Emulab d710 machines and 17 Blue Waters XE compute nodes. However, the real potential of Cassandra in supercomputer is when there is a need for running much larger number of Cassandra nodes. It is reported that there is a production Cassandra cluster which runs over 75,000 nodes with tens of petabytes of data [2]. We

expect that if we run Cassandra in such a massive scale, we will see different characteristics than when we run it in relatively smaller clusters.

5.2. Profiling Request Latency

In order to identify the bottlenecks of the system, we need more detailed information about what consists a request's end-to-end latency. This can be investigated by measuring the decomposition of the requests' latencies by using a latency profiling tool. In Cassandra, an internal profiling tool called *Tracing* exists, which records the progress of a query across the peers while measuring the timestamp of each step. By running this on both environments, we can compare the discrepancies between them, and possibly identify the underlying causes of the system's bottleneck.

5.3. Other Key-Value Stores

We conducted our experiments using Cassandra. However, there are many other distributed key-value stores commonly used in industry that have vastly different characteristics. HBase [12] is a popular open source implementation of Google's BigTable [17]. It shares similar architecture with Cassandra in that it uses SSTables to persist its data. However, HBase is implemented upon a centralized design where meta data is stored differently from the underlying column family's data. It is also notable that HDFS is used for its storage option, and HBase provides better consistency at cost of worse availability.

MongoDB [14] is another popular distributed key-value store. It is most famously known as a document database. Internally, it stores its data in blob, but it maintains B-Tree like index for fast look-ups. Also, when an update request arrives, MongoDB does not persist the change in the underlying database, but stores it in-memory, and only stores a sequential commit log for fault-tolerance. Such design choice of MongoDB and its well defined JavaScript based API led to large user base due to its flexibility and ease of use.

Although we haven't conducted experiments on these distributed key-value stores, we expect to achieve similar results, because the supercomputer would provide the same benefits of extremely fast interconnect and powerful individual compute nodes to other distributed key-value stores as well.

5.4. Optimizations for Supercomputer

We have not attempted to optimize Cassandra for the supercomputer. We did not utilize the technologies specific to the supercomputers such as RDMA and MPI. We expect that porting Cassandra application's network module to use RDMA in place of TCP will result in order of magnitude increase in performance as demonstrated by [20].

6. Conclusions

In this paper, we evaluated the performance of Cassandra on Blue Waters supercomputer in comparison with Cassandra running on Emulab cluster. Although today's supercomputers are mostly used by scientific computations which do not require interactive executions, we claim that it is mainly because of the cost-efficiency reason that most supercomputers are only used in batch applications. The extremely dense and fast interconnect combined with powerful individual compute nodes makes a supercomputer an attractive option for the large internet companies with millions of users. Having a denser and faster interconnect means that the latency would decrease and throughput would increase which are two critical parts of a distributed key-value store's objectives. We analyzed that if the performance of Cassandra on Blue Waters is at least four times better than that running on Emulab, we could claim that Blue Waters is more cost effective in terms of machine cost.

In our first experiment, we measured the tendency of latency and throughput relationship in both environments, and showed that throughput saturates because of a bottleneck in the system, and Blue Waters has much higher saturation throughput than that of Emulab. Next, we showed the scalability of Cassandra in both environments. Both Blue Waters and Emulab scale linearly in terms of throughput, and the latency is stable when the number of nodes increases. We demonstrated that

per-core performance and per-cost throughput is better for Blue Waters with our experimental results. However, we believe that it requires more careful analysis of cost, such as accounting for operation costs. We also showed the read and write latencies are smaller at Blue Waters, whereas the consistency of reads is better at Emulab.

In the future, we plan to experiment with much larger cluster, profile latencies to identify bottlenecks of the system, and compare the results with other distributed key-value stores. Based on the collected results, we will improve the existing distributed key-value store design to run better on the supercomputers.

References

[1] Accumulo. https://accumulo.apache.org. visited on 2015-02-26.

[2] Apple inc.: Cassandra at apple for massive scale. https://www.youtube.com/watch?v=Bc4ql9TDzyg. visited on 2015-04-03.

[3] Blue Waters Hardware Summary. https://bluewaters.ncsa.illinois.edu/hardware-summary. visited on 2015-04-02.

[4] Blue Waters website. https://bluewaters.ncsa.illinois.edu. visited on 2015-04-03.

[5] Cray Gets $188 Million Supercomputer Deal. http://www.forbes.com/fdc/welcome$_mjx.shtml.visitedon2015-04-02$.

[6] Cray supercomputer's cluster compatibility mode(ccm). http://docs.cray.com/books/S-2496-4101/html-S-2496-4101/chapter-9b6qil6d-craigf.html. visited on 2015-04-03.

[7] Dell PowerEdge R730 Price. https://www.dell.com/us/business/p/poweredge-r730/pd. visited on 2015-04-02.

[8] Emulab. https://wiki.emulab.net. visited on 2015-02-26.

[9] Facebook builds its own data center. http://www.datacenterknowledge.com/archives/2010/01/20/facebook-to-build-its-own-data-centers/. visited on 2015-04-03.

[10] Google's data center. http://www.google.com/about/datacenters/. visited on 2015-04-03.

[11] Gordon Supercomputer. http://www.sdsc.edu/us/resources/gordon/. visited on 2015-04-02.

[12] HBase. https://hbase.apache.org. visited on 2015-02-26.

[13] Leveldb. http://leveldb.org. visited on 2015-02-26.

[14] MongoDB. http://www.mongodb.org. visited on 2015-02-26.

[15] P. Bailis, S. Venkataraman, M. J. Franklin, J. M. Hellerstein, and I. Stoica. Probabilistically bounded staleness for practical partial quorums. *Proceedings of the VLDB Endowment*, 5(8):776–787, 2012.

[16] K. Brandstatter, T. Li, X. Zhou, and I. Raicu. Novoht: a lightweight dynamic persistent nosql key/value store. *Under MSST13 review*, 2013.

[17] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber. Bigtable: A distributed storage system for structured data. In *Proceedings of the 7th USENIX Symposium on Operating Systems Design and Implementation - Volume 7*, OSDI '06, pages 15–15, Berkeley, CA, USA, 2006. USENIX Association.

[18] 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.

[19] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels. Dynamo: amazon's highly available key-value store. In *ACM SIGOPS Operating Systems Review*, volume 41, pages 205–220. ACM, 2007.

[20] A. Dragojević, D. Narayanan, O. Hodson, and M. Castro. Farm: Fast remote memory.

[21] S. Gilbert and N. Lynch. Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. *ACM SIGACT News*, 33(2):51–59, 2002.

[22] J. Kepner, W. Arcand, D. Bestor, B. Bergeron, C. Byun, V. Gadepally, M. Hubbell, P. Michaleas, J. Mullen, A. Prout, et al. Achieving 100,000,000 database inserts per second using accumulo and d4m. *arXiv preprint arXiv:1406.4923*, 2014.

[23] A. Lakshman and P. Malik. Cassandra: A decentralized structured storage system. *ACM SIGOPS Operating Systems Review*, 44(2):35–40, Apr. 2010.

[24] F. McSherry, M. Isard, and D. G. Murray. Scalability! but at what cost?

[25] P. ONeil, E. Cheng, D. Gawlick, and E. ONeil. The log-structured merge-tree (lsm-tree). *Acta Informatica*, 33(4):351–385, 1996.

[26] T. Rabl, S. Gómez-Villamor, M. Sadoghi, V. Muntés-Mulero, H.-A. Jacobsen, and S. Mankovskii. Solving big data challenges for enterprise application performance management. *Proceedings of the VLDB Endowment*, 5(12):1724–1735, 2012.

[27] A. Reuther, J. Kepner, W. Arcand, D. Bestor, B. Bergeron, C. Byun, M. Hubbell, P. Michaleas, J. Mullen, A. Prout, et al. Llsupercloud: Sharing hpc systems for diverse rapid prototyping. In *High Performance Extreme Computing Conference (HPEC), 2013 IEEE*, pages 1–6. IEEE, 2013.

[28] B. Schroeder, A. Wierman, and M. Harchol-Balter. Open versus closed: A cautionary tale.

[29] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. *ACM SIGCOMM Computer Communication Review*, 31(4):149–160, 2001.

[30] C. Vaughan, M. Rajan, R. Barrett, D. Doerfler, and K. Pedretti. Investigating the impact of the cielo cray xe6 architecture on scientific application codes. In *Parallel and Distributed Processing Workshops and Phd Forum (IPDPSW), 2011 IEEE International Symposium on*, pages 1831–1837. IEEE, 2011.

[31] W. Wang, L. Xu, and I. Gupta. Scale up vs. scale out in cloud storage and graph processing systems.