Understanding the Causes of Consistency Anomalies in Apache Cassandra

Hua Fan
University of Waterloo
h27fan@uwaterloo.ca

Aditya Ramaraju
University of Waterloo
a2ramaraju@uwaterloo.ca

Marlon McKenzie
University of Waterloo
m2mckenzie@uwaterloo.ca

Wojciech Golab
University of Waterloo
wgolab@uwaterloo.ca

Bernard Wong
University of Waterloo
bernard@uwaterloo.ca

ABSTRACT

A recent paper on benchmarking eventual consistency showed that when a constant workload is applied against Cassandra, the staleness of values returned by read operations exhibits interesting but unexplained variations when plotted against time. In this paper we reproduce this phenomenon and investigate in greater depth the low-level mechanisms that give rise to stale reads. We show that the staleness spikes exhibited by Cassandra are strongly correlated with garbage collection, particularly the "stop-the-world" phase which pauses all application threads in a Java virtual machine. We show experimentally that the staleness spikes can be virtually eliminated by delaying read operations artificially at servers immediately after a garbage collection pause. In our experiments this yields more than a 98% reduction in the number of consistency anomalies that exceed 5ms, and has negligible impact on throughput and latency.

1. OBJECTIVE AND CONTRIBUTIONS

In this paper, we study the staleness of values returned by read operations applied to Cassandra—an open-source distributed storage system that supports eventual consistency using quorum-based replication. More precisely, we investigate the causes of sharp spikes that occur when a constant workload is applied to Cassandra and the staleness metric is plotted as a function of time. Similar anomalies have been observed by Wada et al., Bermbach et al., and Rahman et al., leading to speculation regarding possible causes including side effects of caching and DDoS countermeasures, as well as network jitter [2, 5, 7]. Following the methodology of [5] we run experiments in a single private data center, which isolates the storage system from the effects of caching layers, DDoS countermeasures, and ambient network traffic. None of the mechanisms suggested in prior work account adequately for staleness spikes in such a controlled environment.

Proceedings of the VLDB Endowment, Vol. 8, No. 7
Copyright 2015 VLDB Endowment 2150-8097/15/03.

Our specific contributions with respect to finding the cause of consistency anomalies in Cassandra are as follows:

- First, we reproduce the staleness time series experiment of Rahman et al. (see Figure 3 in [5]) in our data center using a more recent version of Cassandra. Our experiments use an improved consistency metric that is more resilient against clock skew, and hence demonstrate that the staleness spikes are not artifacts caused by the use of one specific metric.
- Next, we formulate the hypothesis that the observed spikes are caused by processing delays, specifically delays due to the Java virtual machine's garbage collection (GC) "stop-the-world" (STW) pause.
- We test our hypothesis experimentally by plotting both the GC pauses and consistency spikes against time. Our results exhibit strong correlations between the times of occurrence of the GC pauses and consistency spikes, as well as between the durations of the GC pauses and the heights of the spikes.
- Guided by insights gained from our experiments we propose a method of removing the staleness spikes by delaying read operations artificially at servers in the time period immediately following a GC pause. This technique yields more than a 98% reduction in the number of consistency anomalies that exceed 5ms, and has negligible impact on throughput and latency.

2. EXPERIMENTS

2.1 Consistency metric

The experiments presented herein use the Γ (gamma) metric for consistency [3], which is similar theoretically to the metric used in [5] but in practice exhibits less noise in environments where clock skew across hosts exceeds operation latencies. The Γ metric quantifies how badly the consistency observed by clients deviates from linearizability [4], which states (informally speaking) that each operation appears to take effect instantaneously at some point between its start and finish times as measured from the perspective of the client who applied the operation. Our experiments measure a fine-grained form of the Γ metric called the *per-value* Γ *score*, which captures deviations from linearizability associated with a collection of operations that access the same key and read or write the same value. Positive Γ scores indicate consistency anomalies, which can be interpreted as stale reads or as write operations that appear to take effect in a non-linearizable order. We plot each Γ score against a point in time, defined relative to the beginning of an experiment, that indicates approximately when the corresponding consistency anomaly occurred. The time values are approximate since anomalies involve the interaction of multiple storage operations that may start and finish at different times.

2.2 Hardware and software environment

We use a private cluster of 11 Intel Xeon E5450 8-core commodity machines with 8GB RAM and 66GB hard disks, connected by Gigabit Ethernet. The hosts use a 64-bit Linux kernel version 2.6.18 and provide Oracle Java 1.7.0u71. Cassandra version 2.0.9 is installed and configured using default parameters except where noted otherwise. We use a modified version of YCSB 0.1.4, similarly to [5], to collect logs of operations from which the consistency metric is computed. One host is used as a coordinator to monitor the experiment and collect logs, five hosts run Cassandra, and up to five other hosts run multi-threaded YCSB clients.

The Oracle HotSpot JVM provides garbage collectors that pause the application threads to evacuate the garbage objects. This pause time is also called "stop-the-world" (STW) time. Even concurrent garbage collectors such as Concurrent Mark Sweep (CMS) include an STW pause in the mark and remark steps. To correlate the inconsistency spikes with GC pause time, we collect the STW start and finish timestamps by parsing the garbage collection logs of the JVMs running Cassandra.

We run the YCSB workload against Cassandra for 120 seconds. Unless specified otherwise, the workload settings are as follows: hotspot distribution with 80% of the load on 20% of the keys, key space size of 500, 32 client threads per YCSB process, and 80%/20% read/write operation mix. We use a skewed distribution to maximize the likelihood of observing consistency anomalies, as in [3, 5]. Cassandra is configured with a replication factor of 3 and consistency level "ONE" is used for both reads and writes, which means that the client waits for only one replica to return an acknowledgment. For each experiment, we present the result of one run, but we repeat the experiment five times to confirm that the pattern observed is reproducible.

2.3 Inconsistency spikes versus STW pause

We reproduce the staleness time series experiment of Rahman et al. (see Figure 3 in [5]) using the Γ metric, and we also plot the time and duration of the GC STW pause as vertical dotted lines in the same graph, shown in Figure 1. The position of these lines on the x-axis indicates the time of an STW pause, and the height of these lines on the y-axis indicates the duration of the pause. Both GC STW pause times and Γ metrics are aggregate results from all hosts. For the sake of clarity STW times less than 5 ms are not plotted. The spike pattern in Figure 1 at around 40 seconds is very similar to Figure 3 in [5] at around 55 seconds.

Informally speaking, the spikes demonstrate a strong correlation with the GC pause. In terms of experiment time the spikes align precisely with the STW pauses indicated by the dotted lines perpendicular to the x-axis. Furthermore, the height of the spikes corresponds closely to the length of the GC pause, which is indicated by the height of the dot-

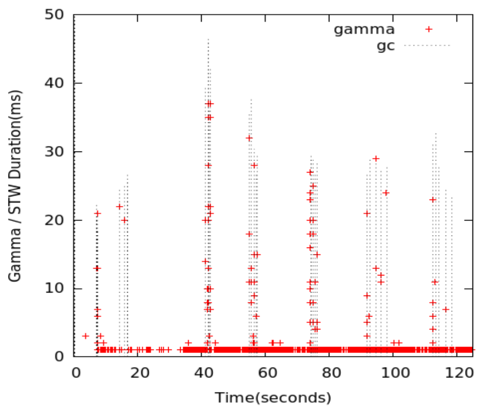


Figure 1: Time series of Γ score and GC pause time with five YCSB hosts and replication factor three.

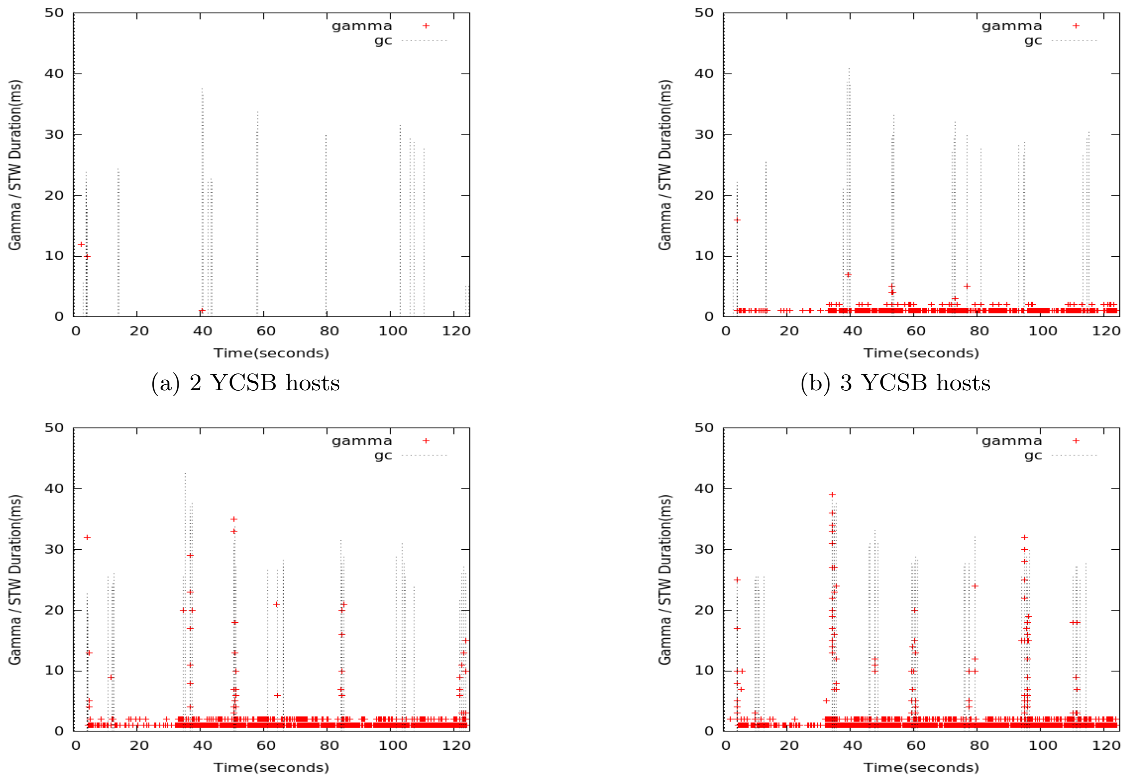
ted lines. In other words, nearly all consistency violations happen at a time near a GC STW pause, and most of the observed Γ scores, which quantify the severity of a consistency violation, are less than the duration of the pause.

To explore the internal cause of the inconsistency spikes, we run the experiments with a fixed number of Cassandra hosts but various number of YCSB hosts. In this experiment the replication factor is set to 5 to allow each Cassandra host to hold one replica. Figure 2 shows the results using 2-5 YCSB hosts and 5 Cassandra hosts. As the number of YCSB hosts is varied each YCSB host has similar throughput, and the aggregate throughput is proportional to the number of YCSB hosts. We observe that as the number of YCSB hosts increases the consistency violations become more severe but generally do not exceed the length of the STW pause—an observation we exploit later on in Section 2.4. Furthermore, the number of spikes and the number of positive Γ scores increases more than linearly with the number of YCSB hosts (e.g., compare Figure 2 (a) with (c)). Thus, consistency anomalies occur readily as the offered load increases, and may be of concern in practice even when the storage system is operating at less than full throttle.

We propose that the GC STW pause is the cause of the consistency anomalies and the underlying rationale is as follows. Recall that a value returned by a read is stale if that value is not the most recent value that was written to Cassandra, meaning that a subset of the replicas did not receive the last updated value in time for the read to see it. When the JVM of some replica experiences the GC STW pause, all application threads are stopped and that delays the processing of updates at this replica, while the same updates may be applied successfully at other replicas. After application threads resume from STW, any reads from this replica return a stale value until the backlogged updates are applied. Depending on how long it takes to clear the backlog, reads may in theory return values that are stale by more than the duration of the STW pause time at a replica, but we rarely observe this in our experiments.

2.4 Smoothing the inconsistency spikes

In this subsection, we investigate the technique of smoothing out the inconsistency spikes by delaying reads artificially during a short interval of time following a GC STW pause, which we call the *delay period*. The delay period begins as



(a) 2 YCSB hosts (b) 3 YCSB hosts

(c) 4 YCSB hosts (d) 5 YCSB hosts

Figure 2: Time series of Γ score and GC pause time with replication factor five.

soon as the GC STW pause is detected, and lasts for a duration that depends on the method used to detect the pause. Operation delays are governed by the following policy: inside the delay period read operations are stalled until the delay period is finished, whereas write operations follow the usual execution path; outside the delay period reads and writes both follow the usual execution path.

We experimented with two concurrent garbage collectors: ConcurrentMarkSweep (CMS), the default GC, and Garbage First (G1), a newer GC provided by the HotSpot JVM. The results for both garbage collectors are similar, and so we only present the results for CMS.

We use two mechanisms to detect the GC STW pause: *GC-notification* and *Free-heapsize*. Since Java 7 update 4, an API called *GarbageCollectionNotificationInfo* is available for receiving notifications from the GarbageCollectorMXBean. It can provide the start time and duration of the GC, which we use to define the delay period. Once a GC occurs, the delay period is activated in the notification handler for the GC duration specified by the management bean. We refer to this method as *GC-notification*. The *Free-heapsize* method instead traces free heap size in the read operation routine. Once a large free heap size increase ($\geq$100MB in our setting) is detected, we consider it a sign of garbage collection occurring. In that case the delay period is activated for a fixed period of 50ms—an empirically determined upper bound on the length of the STW pause in our experiments. As noted earlier in Section 2.3 the length of the STW pause is, in turn, an approximate upper bound on the Γ scores observed during inconsistency spikes.

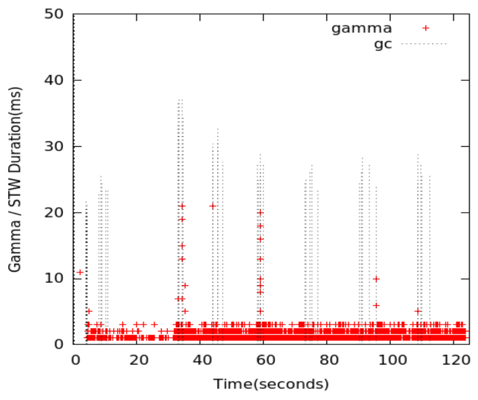
Figure 3 (a) shows the result of using the GC-notification method with a replication factor of five and five YCSB hosts. In terms of the number of positive Γ scores and the height of the spikes, the consistency anomalies are less severe than in Figure 2 (d), which uses the same system and workload parameters. But there are still three spikes in excess of 20ms in height. Thus, the smoothing is incomplete, possibly because the GarbageCollection notification handler is not guaranteed to execute in a timely fashion. In other words, the execution of the notification handler that activates the delay period may lag behind the end of the STW pause.

Figure 3 (b) shows the results using the Free-heapsize method. We find that the spikes are nearly removed and 98.5% Γ scores more than 5ms are also removed. With the exception of one Γ score of around 6ms at the beginning of the experiment, and one Γ score of around 15ms at the 108th second, all the remaining points are under 5ms.

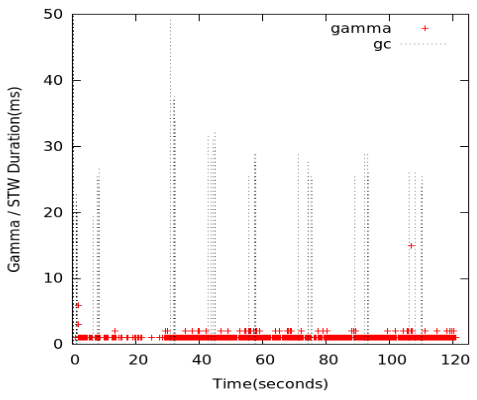
Table 1 shows the throughput and read latency influence of the artificial read delays. GC notification has no significant drop in throughput, 1% increase in average read latency, no significant increase in max read latency, and 0.6ms increase in 95%-ile read latency. Free heapsize has no significant drop in throughput, 1.6% increase in average read latency, no significant increase in max read latency, and 0.8ms increase in 95%-ile read latency. Thus, our results show that the spikes can be virtually eliminated with very little overhead in terms of throughput and latency.

In future work we plan to explore the following ideas:

- *Combination of GC-notification and Free-heapsize.* In hosts with larger main memories, the GC pause time



(a) GC-notification method



(b) Free-heapsize method

Figure 3: Smoothing of inconsistency spikes by delaying reads artificially after a GC STW pause.

	#violation ($\Gamma > 5$ms)	Aggregate Throughput (ops/s) ±standard error	Average Latency(ms) ±standard error	Max Latency (ms) ±standard error	95%-ile Latency (ms) ±standard error
No delay	133	8604 ± 10	9.23 ± 0.02	324 ± 11	25.0 ± 0
GC notification	31	8584 ± 21	9.33 ± 0.03	311 ± 4	25.6 ± 0.2
Free heapsize	2	8586 ± 15	9.38 ± 0.03	322 ± 13	25.8 ± 0.2

Table 1: Comparison of consistency violation, throughput and read latency under read delay

may on occasion be much longer than in our experiment, which could require an excessively long fixed delay period duration in Free-heapsize. On the other hand, the notification lag of the GC notification method is usually quite small. Combining the two strategies could provide both agile reaction and a precise calculation of the delay period duration.

- *Garbage collection tuning.* As the consistency spikes result from the GC pause, ordinary GC tuning techniques may reduce the spikes by shortening the STW pause time. For example, one could tune the size of the young generation in heap. Real-time JVMs such as C4 [6], or (nearly) pauseless garbage collectors such as Shenandoah [1], may also be helpful.
- *Controlling internal activities in the storage system.* Consistency anomalies could be made more predictable to avoid inconsistency at inconvenient times, by means of explicitly invoking GC, or controlling internal activities that may lead to GC. For example, in Cassandra such internal activities include the *flush*, also referred as *minor compaction*, which flushes in-memory data from the *memtable* to an *SSTable* file on disk, and may be accompanied by a substantial GC since a large chunk of memory becomes available for recycling.

3. CONCLUSION

Our experiments reproduce the inconsistency spikes observed in [5] and demonstrate that the stop-the-world pause during garbage collection is the cause of these spikes. Specifically, our results show a strong correlation between the GC pauses and inconsistency spikes in terms of both the time of occurrence and the GC duration vs. spike magnitude. We explain this phenomenon by observing that the GC STW pause causes updates to be applied at different times at various replicas. Furthermore, we demonstrate two methods of delaying read operations artificially to smooth out the spikes: GC-notification and Free-heapsize. Both methods incur only a slight overhead in throughput and read latency.

4. ACKNOWLEDGMENTS

Authors supported in part by the Natural Sciences and Engineering Research Council of Canada (NSERC).

5. REFERENCES

[1] Shenandoah. http://icedtea.classpath.org/wiki/Shenandoah.

[2] D. Bermbach and S. Tai. Eventual consistency: How soon is eventual? An evaluation of Amazon S3's consistency behavior. In *Proc. Workshop on Middleware for Service Oriented Computing (MW4SOC)*, 2011.

[3] W. Golab, M. R. Rahman, A. AuYoung, K. Keeton, and I. Gupta. Client-centric benchmarking of eventual consistency for cloud storage systems. In *Proc. of the 34th International Conference on Distributed Computing Systems*, pages 493–502, 2014.

[4] M. P. Herlihy and J. M. Wing. Linearizability: A correctness condition for concurrent objects. *ACM Trans. Program. Lang. Syst.*, 12(3):463–492, 1990.

[5] M. R. Rahman, W. Golab, A. AuYoung, K. Keeton, and J. J. Wylie. Toward a principled framework for benchmarking consistency. In *Proc. USENIX Workshop on Hot Topics in System Dependability (HotDep)*, 2012.

[6] G. Tene, B. Iyengar, and M. Wolf. C4: The continuously concurrent compacting collector. *SIGPLAN Not.*, 46(11):79–88, June 2011.

[7] H. Wada, A. Fekete, L. Zhao, K. Lee, and A. Liu. Data consistency properties and the trade-offs in commercial cloud storage: the consumers' perspective. In *Proc. Conference on Innovative Data Systems Research (CIDR)*, pages 134–143, 2011.