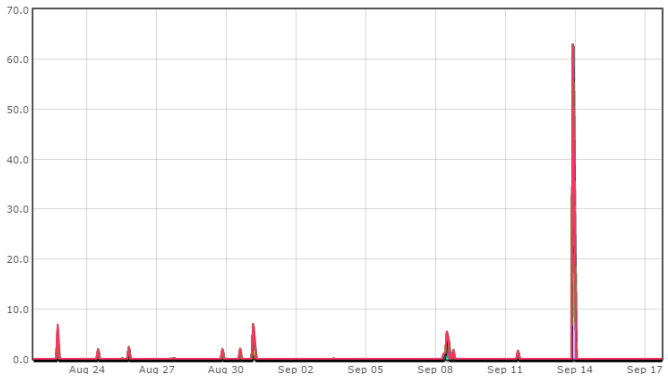

Figure 10: Events Dropped Per Second over 1 Month

second and all such issues are resolved within approximately 10 seconds. The largest spike of 62 events per second was caused by a hardware failure and bug which caused retries to be incorrectly attempted; this issue was caused merely by implementation, not a design flaw.

10. CONCLUSIONS

Overall, the New CRS improves or maintains data reliability, data availaibility and query responsiveness, while reducing cpu and memory usage.

10.1 Data Integrity

With the Legacy CRS, there were numerous incidents where we lost millions of of session events and tens of thousands of crash reports. While the Legacy CRS lacked the requisite monitoring to get exact figures, it would routinely lose millions of session events and hundreds of thousands of crash reports. With the new CRS, there is still occasional data loss; however it is mitigated to very small periods of time due to temporary errors that exceed the duration of capture service retries. Furthermore, none of that data is permanently lost, it is merely lost from the near-real time reporting of the CRS and is still reprocessable. This is very clearly an improvement over the Legacy CRS.

10.2 Latency

Next we can compare the query latency and end-to-end persistence latency of the Legacy and New CRS. The end-to-end latency is the total time elapsed between report capture and data being visible to the reporter api. For the Legacy CRS, it is on the order of 1 to 2 seconds in the normal case. The new CRS achieves a steady 550 ms average and 900 ms 95th percentile end-to-end latency. During load spikes, the end-to-end latency will rise if the provisioned consumer processes cannot meet the incoming crash rate. This is the tradeoff that we have made to ensure that data is eventually persisted[2].

For query latency, the load testing section above clearly demonstrates the general superiority of New CRS. The most common scenario in production is around 10 concurrent user queries, which average 20 to 100 reports in size; the New CRS responds to those queries with a 54% decrease in latency. In the case of very large queries, the New CRS did perform slightly worse with a 3% increase in query latency. However, the New CRS is arbitrarily horiztonally scalable and thus such poorly performing queries can be improved simply by splitting them into smaller concurrent queries.

10.3 Resource Usage

As mentioned above, the new CRS needs only provision 6 m3.xlarge instances and 1 m1.xlarge instance, thanks to amortizing the cost of high availability across other services. The Legacy CRS used 8 collection servers, 3 MySQL master-slave pairs and a single reporter server. All of these had varying specifications, which were invariably larger than the m1 and m3 instances provisioned in AWS. In total, the Legacy system used 208 CPUs and 710 GB of RAM: 48 cores and 100GB for collection; 144 cores and 564 GB for MySQL; and 16 cores and 46GB for reporting. The New CRS uses 80 cores and 308 GB of RAM: 8 cores and 30 GB for the capture process; 40 cores and 154 GB for data persistence; and 4 cores and 15 GB for reporting. This represents a 63% decrease in CPU usage and a 59% decrease in memory usage.

10.3.1 Other Costs

While it is less quantitative, the time spent to maintain these systems should also be taken into account. The New CRS is built on cloud infrastructure that survives without manual interaction even in the event of node failure. This also allows for much quicker turn around on server maintenance, scaling and configuration. Furthermore, we have found couchbase to be a very stable system that requires little configuration or headache to maintain, as long as its N1QL indexing is not enabled. It is difficult to put a price on these advantages, but they have certainly allowed us more rapid and reliable testing and provisioning of hardware than if we were still using a proprietary data center.

11. FUTURE WORK

At the moment, the reporting API is very bare and can only optimize queries based on time ranges. We would like to continue to investigate other methods for asynchronous indexing, either by building an in-house system using rdbms for index storage (as we have primitively implemented here) or by taking advantage of open source software such as Lucene and Elasticsearch. For either of these approaches our primary concerns are focused on enabling full-text search and creating an index service that scales both with the amount of incoming data and the number of index scan requests. Furthermore, there are more demands from game developers to increase payload size by including even more detailed stack traces, higher resolution images and/or short video clips. We seek to support these changes by evaluating the impact of storing even larger media files on Couchbase and how best to efficiently process such bulky reports, at protocol and infrastructure levels.

[2]For the current provisioning, a 5 minute crash spike reaching 3300 reports per second (over a 10X rate spike) will incur a worst case latency no more than 5 minutes.

12. REFERENCES

[1] F. 7. Benchmarking network performance of m1 and m3 instances, 2014.

[2] Altoros. The nosql technical comparison report, 2014.

[3] I. Amazon.com. Amazon web services, 2016.

[4] Apache. Apache kafka, 2016.

[5] S. Bisbee. Scale it to billions - what they don't tell you in the cassandra readme, 2015.

[6] E. Co. Elasticsearch, 2016.

[7] E. P. Corporation. Benchmarking top nosql databases, 2015.

[8] Couchbase. Couchbase, 2016.

[9] Y. Dang, R. Wu, H. Zhang, D. Zhang, and P. Nobel. Rebucket: A method for clustering duplicate crash reports based on call stack similarity. In *Proceedings of the 34th International Conference on Software Engineering*, ICSE '12, pages 1084–1093, Piscataway, NJ, USA, 2012. IEEE Press.

[10] Datastax. Cassandra configuration, 2016.

[11] Datastax. Getting started with time series data modeling, 2016.

[12] J. Klein, I. Gorton, N. Ernst, P. Donohoe, K. Pham, and C. Matser. Performance evaluation of nosql databases: A case study. In *Proceedings of the 1st Workshop on Performance Analysis of Big Data Systems*, PABS '15, pages 5–10, New York, NY, USA, 2015. ACM.

[13] I. B. Times. Battlefield 4: Bugs and issues lead to backlash, 2013.

[14] R. Wu. Diagnose crashing faults on production software. In *Proceedings of the 22Nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, FSE 2014, pages 771–774, New York, NY, USA, 2014. ACM.