Heisenberg Was on the Write Track

Pat Helland
Salesforce
PHelland@Salesforce.com

Abstract

This paper argues that in a distributed system comprised of storage on multiple servers, you can know *where* you write the replicas of your data or you can know *when* the replicas will be written but you can't know both.

Introduction

Consider a cluster in which there are a number of simple commodity servers with attached storage and a network connecting the servers. Each server has a noticeable failure rate but the cluster tolerates these failures. Typically, data is replicated across three of these servers and the failure of a server causes the creation of a third replica on yet another server.

A committed write is typically defined as one that's durable on three servers. In many systems, this is accomplished with a write to a file system like GFS [3] or HDFS [4] with a NameNode controlling placement and three DataNodes holding a large block of the file. Coping with a DataNode failure requires the help of the NameNode. Some systems like Dynamo [2] will write to three replicas but the specific three replicas can vary based on circumstances such as node failure. In Dynamo, there is no master or NameNode. GFS and HDFS have strongly consistent placement of data over replicas. Dynamo is weakly consistent.

Many systems count on log-write latency of group commit buffers to be consistent and fast. An SLA under 5ms 99.9% of the time is typical when humans await the commit. This is best done with weakly consistent placement of data over replicas.

Writing to Multiple Replicas

Say you're writing a log to three replicas picked by the NameNode. This is fast only if all three are fast. If one or more of the preselected replicas is a laggard, you can delay and annoy the humans awaiting commit. Maybe there's a failure of a replica and the NameNode gets involved. Odds are very high that you've missed your SLA. If the NameNode is delayed, it's even worse!

When writing to three prescribed locations, you may be delayed.

Contrast this to a scheme where the writer can retry the log write to another server. The writer may launch three writes to the preferred replicas and, if one or more take too long, retry to yet another replica. This is very similar to the latency bounding technique described in the excellent paper "The Tail at Scale" [1]. What's different is that this is not a read and, hence, we need to deal with the log write landing on a different server than one of the preferred replicas.

In other words, we don't quite know *where* we write but we can construct much stronger bounds on *when* the write happens.

7th Biennial Conference on Innovative Data Systems Research (CIDR '15) January 4-7, 2015, Asilomar, California, USA.

Correlated Stalls Really Mess Up Your SLAs

When all your target replicas can get stalled at the same time, it's hard to have tight SLAs. You need to look at anything that affects all your servers at the same time. For example:

- **Network:** If the network is saturated and prevents timely communication to your storage replicas that will wreak havoc on your SLA to commit the log write.
- **Strongly Consistent Replica Coordinator:** Whether it's a single process or some Paxos-like thing, if it's strongly consistent, it can stall and impact the routing of writes to ALL the replicas. *Strongly consistent placement goes hand-in-hand with correlated stalls.*

It's one thing to have a stall on a single replica. That can be worked around. Stalling on *all* the storage replicas will impact the SLA as shown to the user.

Dealing with Uncorrelated Stalls of Replicas

There are a couple of techniques for dealing with replica stalls:

- **Love the Ones You're With:** If one or more of the writes to the preferred replicas stall and are not confirmed, continue trying different servers until you're satisfied. If managed correctly, you can keep statistically tight SLAs while sometimes landing the writes in auxiliary replicas.
- **Two Outta Three Ain't Bad:** When 2 replicas are durable and we are actively creating a third, this can meet our data availability requirements. The log writer can launch three writes and respond "commit" to the user when two writes are durable. Combine this with an active retry for the third replica (and replica repair after a crash) and we meet our durability goals.

Conclusion

Batch systems like GFS [3] and HDFS [4] are very important in our computing landscape. They provide huge throughput using interesting placement techniques for reliable data. This can, however, come at the expense of predictable and short SLAs.

Fluctuating SLAs are a much bigger problem for user facing OLTP style updates to data. Tighter SLAs mean the actual placement of the replicas of the data is only loosely known.

References

[1] Dean, J.; Barroso, L. A. (2013) " The Tail at Scale". Communications of the ACM, Vol. 56 No.2, Feb 2013, P.74

[2] Decandia, G.; Hastorun, D.; Jampani, M.; Kakulapati, G.; Lakshman, A.; Pilchin, A.; Sivasubramanian, S.; Vosshall, P.; Vogels, W. (2007). "Dynamo" Proceedings of the Twenty-First ACM Symp on Operating Systems Principles – SOSP '07. P.205.

[3] Ghemawat, S.; Gobioff, H.; Leung, S. T. (2003) "The Google File System". *Proceedings of the Nineteenth ACM Symp on Operating Systems Principles - SOSP '03*. P. 29.

[4] "HDFS Architecture Guide". http://hadoop.apache.org/docs/r1.2.1/hdfs_design.html