BLOG@CACM

The *Communications* Web site, http://cacm.acm.org, features more than a dozen bloggers in the BLOG@CACM community. In each issue of *Communications*, we'll publish selected posts or excerpts.

twitter
Follow us on Twitter at http://twitter.com/blogCACM

DOI:10.1145/1831407.1831411 http://cacm.acm.org/blogs/blog-cacm

In Search of Database Consistency

Michael Stonebraker discusses the implications of the CAP theorem on database management system applications that span multiple processing sites.

Michael Stonebraker
"Errors in Database Systems, Eventual Consistency, and the CAP Theorem"
http://cacm.acm.org/blogs/blog-cacm/83396

Recently, there has been considerable renewed interest in the CAP theorem[1] for database management system (DBMS) applications that span multiple processing sites. In brief, this theorem states that there are three interesting properties that could be desired by DBMS applications:

C: Consistency. The goal is to allow multisite transactions to have the familiar all-or-nothing semantics, commonly supported by commercial DBMSs. In addition, when replicas are supported, one would want the replicas to always have consistent states.

A: Availability. The goal is to support a DBMS that is always up. In other words, when a failure occurs, the system should keep going, switching over to a replica, if required. This feature was popularized by Tandem Computers more than 20 years ago.

P: Partition-tolerance. If there is a network failure that splits the processing nodes into two groups that cannot talk to each other, then the goal would be to allow processing to continue in both subgroups.

The CAP theorem is a negative result that says you cannot simultaneously achieve all three goals in the presence of errors. Hence, you must pick one objective to give up.

In the NoSQL community, the CAP theorem has been used as the justification for giving up consistency. Since most NoSQL systems typically disallow transactions that cross a node boundary, then consistency applies only to replicas. Therefore, the CAP theorem is used to justify giving up consistent replicas, replacing this goal with "eventual consistency." With this relaxed notion, one only guarantees that all replicas will converge to the same state eventually, i.e., when network connectivity has been reestablished and enough subsequent time has elapsed for replica cleanup. The justification for giving up C is so that the A and P can be preserved.

The purpose of this blog post is to assert that the above analysis is suspect, and that recovery from errors has more dimensions to consider. We assume a typical hardware model of a collection of local processing and storage nodes assembled into a cluster using LAN networking. The clusters, in turn, are wired together using WAN networking.

Let's start with a discussion of what causes errors in databases. The following is at least a partial list:

1. Application errors. The application performed one or more incorrect updates. Generally, this is not discovered for minutes to hours thereafter. The database must be backed up to a point before the offending transaction(s), and subsequent activity redone.

2. Repeatable DBMS errors. The DBMS crashed at a processing node. Executing the same transaction on a processing node with a replica will cause the backup to crash. These errors have been termed "Bohr bugs."[2]

3. Unrepeatable DBMS errors. The database crashed, but a replica is likely to be ok. These are often caused by weird corner cases dealing with asynchronous operations, and have been termed "Heisenbugs."[2]

4. Operating system errors. The OS crashed at a node, generating the "blue screen of death."

5. A hardware failure in a local cluster. These include memory failures, disk failures, etc. Generally, these cause a "panic stop" by the OS or the DBMS. However, sometimes these failures appear as Heisenbugs.

6. A network partition in a local cluster. The LAN failed and the nodes

can no longer all communicate with each other.

7. A disaster. The local cluster is wiped out by a flood, earthquake, etc. The cluster no longer exists.

8. A network failure in the WAN connecting the clusters together. The WAN failed and clusters can no longer all communicate with each other.

First, note that errors 1 and 2 will cause problems with any high availability scheme. In these two scenarios, there is no way to keep going; i.e., availability is impossible to achieve. Also, replica consistency is meaningless; the current DBMS state is simply wrong. Error 7 will only be recoverable if a local transaction is only committed after the assurance that the transaction has been received by another WAN-connected cluster. Few application builders are willing to accept this kind of latency. Hence, eventual consistency cannot be guaranteed, because a transaction may be completely lost if a disaster occurs at a local cluster before the transaction has been successfully forwarded elsewhere. Put differently, the application designer chooses to suffer data loss when a rare event occurs, because the performance penalty for avoiding it is too high.

As such, errors 1, 2, and 7 are examples of cases for which the CAP theorem simply does not apply. Any real system must be prepared to deal with recovery in these cases. The CAP theorem cannot be appealed to for guidance.

Let us now turn to cases where the CAP theorem might apply. Consider error 6 where a LAN partitions. In my experience, this is exceedingly rare, especially if one replicates the LAN (as Tandem did). Considering local failures (3, 4, 5, and 6), the overwhelming majority cause a single node to fail, which is a degenerate case of a network partition that is easily survived by lots of algorithms. Hence, in my opinion, one is much better off giving up P rather than sacrificing C. (In a LAN environment, I think one should choose CA rather than AP.) Newer SQL OLTP systems appear to do exactly this.

Next, consider error 8, a partition in a WAN network. There is enough redundancy engineered into today's WANs that a partition is quite rare. My experience is that local failures and application errors are way more likely.

"In the NoSQL community, the CAP theorem has been used as the justification for giving up consistency."

Moreover, the most likely WAN failure is to separate a small portion of the network from the majority. In this case, the majority can continue with straightforward algorithms, and only the small portion must block. Hence, it seems unwise to give up consistency all the time in exchange for availability of a small subset of the nodes in a fairly rare scenario.

Lastly, consider a slowdown either in the OS, the DBMS, or the network manager. This may be caused by a skew in load, buffer pool issues, or innumerable other reasons. The only decision one can make in these scenarios is to "fail" the offending component; i.e., turn the slow response time into a failure of one of the cases mentioned earlier. In my opinion, this is almost always a bad thing to do. One simply pushes the problem somewhere else and adds a noticeable processing load to deal with the subsequent recovery. Also, such problems invariably occur under a heavy load—dealing with this by subtracting hardware is going in the wrong direction.

Obviously, one should write software that can deal with load spikes without failing; for example, by shedding load or operating in a degraded mode. Also, good monitoring software will help identify such problems early, since the real solution is to add more capacity. Lastly, self-reconfiguring software that can absorb additional resources quickly is obviously a good idea.

In summary, one should not throw out the C so quickly, since there are real error scenarios where CAP does not apply and it seems like a bad tradeoff in many of the other situations.

References

1. Eric Brewer, "Towards Robust Distributed Systems," http://www.cs.berkeley.edu/~brewer/cs262b-2004/PODC-keynote.pdf
2. Jim Gray, "Why Do Computers Stop and What Can be Done About It," Tandem Computers Technical Report 85.7, Cupertino, CA, 1985. http://www.hpl.hp.com/techreports/tandem/TR-85.7.pdf

Disclosure: Michael Stonebraker is associated with four startups that are producers or consumers of database technology.

Readers' comments

"Degenerate network partitions" is a very good point—in practice I have found that most network partitions in the real world are of this class.

I like to term certain classes of network partitions "trivial." If there are no clients in the partitioned region, or if there are servers in the partitioned region, it is then trivial. So it could involve more than one machine, but it is then readily handled.

—Dwight Merriman

I think a lot of the discussion about distributed database semantics, much like a lot of the discussion about SQL vs. NoSQL, has been somewhat clouded by a shortage of pragmatism. So an analysis of the CAP theorem in terms of actual practical situations is a welcome change :-)

My company, GenieDB, has developed a replicated database engine that provides "AP" semantics, then developed a "consistency buffer" that provides a consistent view of the database as long as there are no server or network failures; then providing a degraded service, with some fraction of the records in the database becoming "eventually consistent" while the rest remain "immediately consistent." Providing a degraded service rather than no service is a good thing, as it reduces the cost of developing applications that use a distributed database compared to existing solutions, but that is not something that somebody too blinded by the CAP theorem might consider!

In a similar vein, we've provided both NoSQL and SQL interfaces to our database, with different trade-offs available in both, and both can be used at once on the same data. People need to stop fighting over X vs. Y and think about how to combine the best of both in practical ways!

—Alaric Snell-Pym

Michael Stonebraker is an adjunct professor at the Massachusetts Institute of Technology.