CAP TWELVE YEARS LATER: HOW THE “RULES” HAVE CHANGED

Publication Setting

The article this workshop report is written about was published in February 2012 in IEEE Computer Society's journal “Computer”. Two things make it very interesting: First, it is a very recent publication and less than one year old. Second and more important, it was written by Eric Brewer himself, who originally formulated the CAP theorem twelve years ago. CAP being one of the most important foundations of Distributed Information Systems and today's Cloud Computing services, it is quite interesting what Brewer himself thinks of the past development, and how he reflects about CAP and what it started in retrospective.

Content

Recapitulation of ACID, BASE and CAP

Brewer starts his article by restating how he originally formulated the CAP theorem. In a distributed information system, only two out of three properties can be guaranteed at the same time:

- **Consistency**, meaning that data is the same on every site of the system. It acts as if data is stored on a single system, making it indifferent where and when you request a data item; one always gets the “correct” information.

- **Availability**, a property which is about how often or not a system can be reached. For example, Two Phase Commit systems are not available in between the “prepare-to-commit” and actual “commit” steps, because there is no final consistent state yet.

- **Partition Tolerance**, which concerns a system's resilience against loss of communication between sub systems. The question is whether or not the system still behaves in a predictable way in case of malfunctioning connectivity.

He continues to explain that initially, people began to seek tradeoff between Consistency and Availability since partition tolerance was being regarded as fixed since network partitioning happens and can not be prevented. And while traditional ACID (Atomicity, Consistency, Isolation, Durability) database systems are on the extreme consistency end forfeiting availability, recent NoSQL and BASE (Basically Available, Soft state, Eventually consistent) systems tend do go more into the availability direction, risking consistency.
Corrections

Directly after reiterating the common CAP formulation, Brewer immediately points out what he now regards wrong with how he originally phrased it: “Two out of three” is an over simplification. While the general idea is in fact correct, it is not as easy as simply accepting the rule and choosing to have either Availability or Consistency when building a database system. Two things are then pointed out.

**CAP is not binary**

None of the three CAP properties are binary. Availability of course can be scaled from zero to a hundred percent, and consistency has many nuances in between “consistent” and “inconsistent”. For example, there is the concept of eventual consistency which temporarily might violate consistency, but guarantees that the system will converge against a consistent state over time. Other applications choose master and slave copies to guarantee always being able to extract a consistent state from the system.

Existing systems have recognized this and are exploring the space between consistency and availability, and many examples are named. Facebook by default serves data to users from remote copies that do not always necessarily have the most recent data but are geographically nearby the user in order to guarantee a high degree of availability, but redirects users to the master copy on writing processes and keeps users connected to it for several seconds in order to keep consistency at the same time. The idea behind Yahoo! PNUTS is somewhat similar.

Similarly but not as well investigated, network partitioning has different degrees as well. Since by concept, there is no global view of a network, there is also no global view of network partitioning. So while one slave node named A might detect a partition and cannot reach the master node B, B in return does not care about A and never notices any kind of partition.

Another aspect is the relationship between latency and partition. As it is now, a partition is simply an infinite latency – or in reality, a timeout that has to be chosen somehow. This also leaves room for research.

**The partition decision**

Contrary to what one might understand, CAP does not imply a one time decision. There is no need to choose consistency or availability or anything in between beforehand. As a matter of fact, a system can guarantee both availability and consistency when fully connected. The decision needs to be made not until the moment a partition happens. Brewer hence calls this the **partition decision**.

Furthermore, this decision then can be made at very fine granularity, multiple times at different points in time and for different actions and data. A system can take many things into account when operating while being partitioned: Time, current state, which data is being operated on and so on. And depending on those, the way to proceed can be planned.

To sum it up, CAP is not the “Two out of three” triangle wherein a system chooses its position once, but a finely granular space in which developers can move around continuously at run time.
Partition Managing

Building on those two realizations of CAP not being binary and the fine granularity of the partition decision, Brewer proposes a simple model to enable said moving around in CAP space. First thing, a system requires a partition detection mechanism in order to be able to react to situations in which a decision is needed. The decision itself is then implied by entering a so called partition mode which has special restrictions or keeps more information about state changes. When the connection between subsystems is finally reestablished, actions and information from partition mode then can be used to construct a newly merged, hopefully consistent state in what is dubbed the partition recovery step.

Partition Mode

Many things that can be done in partition mode are explained. The most crucial aspect Brewer mentions are so called invariants that need to be kept by the system. A simple example for such an invariant is that in a database system, primary keys need to be unique. While fully consistent systems like 2PC automatically keep such constraints intact as long as they are somehow included in the data model, other systems might need to pay extra attention to such invariants during partition mode.

A system designer needs to think of all such invariants that need to be kept and look at the cross product of each invariant and all the possible actions that might affect them. Those actions that might violate invariants in partition mode then have to be adjusted in said partition mode: First, they can be prohibited taking a big step towards consistency and forfeiting availability. Second, additional data can be kept, for example history of who did something and when instead of only tracking the current state, quite similar to a version control system commit history. Last, actions can be modified in another way like delaying the actual data transmission to a later point in time when connection is reestablished.

Combinations are of course possible: An ATM could risk the invariant “an account balance cannot fall below zero” when partitioned away from the central database and nevertheless permit withdrawals of up to $100, risking consistency in favor to availability. On the other hand, anything above $100 could be permitted, keeping the risk for inconsistency small and at the same time offer availability. The critical amount of money can be of course freely scaled, giving easy control over the exact position in between A and C within the CAP space.

To give another example, regarding the earlier mentioned primary key invariant, when introducing a new primary key in partition mode, a system might record the origin and timestamp of creation of that new primary key. That way, when recovering later on, the system could be able to distinguish two conflicting entries with identical primary keys, renaming one of them and restoring consistency and the invariant.

Partition Recovery

When finally the partition ends, Brewer requires a system to do a so called partition recovery step. The purpose of that step is to merge the state of the two partitioned systems into a consistent state and maybe repair any fault that might have happened during partition mode.

The first possibility that might happen is that automatic merging is possible. This might be done using the additional operation history and action timestamps, by ordering actions that happened on both sides. All actions then can be re-applied to the last known consistent state before the partition happened to create a new consistent state. In case of detected concurrency or conflict of actions,
commutative operations are needed, maybe by having modified actions in partition mode in a commutative way. An example for commutative operation would be the addition of text to a document, as the ordering of such operations is not important. So the editing action of a document could be modified to only allow adding text. In that context, CRDTs (Commutative Replicated Data Types) are mentioned that help maintaining commutative operations and data merging.

If such is not possible, human escalation might be necessary. Mistakes that have been made need to be corrected; especially so-called externalized operations need to be analyzed, checked for errors and compensated. Externalized operations are actions that give out information to other systems or users that are not in control of the original system. An example for such externalized operations could be found in an airplane booking system that allows booking seats while in partition mode, confirming the users to have booked a seat. While this might work as long as the flight is not full, in case there is an overbooking, the system has to be able to tell such events when merging again, or in other words, keeping history is necessary. Having this kind of information, a human then can solve such a conflict by refunding one customer or finding any other sort compromise. Of course compensating errors is not the same as not doing them in the first place and is also a possible subject to research.

Conclusion

Brewer concludes his article by giving advice to system designers. They should look at their system closely and look at each invariant and carefully design the system to react on network partition. The Freedom inside the CAP space should be explored, as the best systems are those that pay the most attention to such invariant details. He formulates this very concisely:

“System Designers should not blindly sacrifice consistency or availability [...]. They can optimize both properties through careful management of invariants during partitions.”

Discussion & Personal Thoughts

As already mentioned, the publication is very interesting as it is written by the inventor of CAP itself. Although it does not contain any groundbreaking new theory, it nicely sums up all the possibilities and technologies that have been researched over the last decade, boiling everything down to a unified model. Brewer also succeeds to bring in many examples of already existing solutions, setting the readers way of thinking into his model – personally, before reading the article I never thought about Google Docs and HTML5 local storage having anything to do with CAP. He also always emphasizes not to think about the restrictions that CAP gives but consider the possibilities and freedom a system designer has, as CAP allows for very fine grained decisions, opposed to the initial binary formulation of the theorem.

The discussion that came up after the workshop presentation was all about described freedom in CAP. The feasibility of examples like the airplane booking system was questioned, but in the end everything resulted in the realization of the high degree of dynamics Brewers model allows. When facing problems the way a system is implemented, things that happen in partition mode can be slightly adjusted or more factors can be included in the partition decision. For example, local services can take daytime into account, making the system more available at day while consistency can be more important at night when customers sleep. This also coincides with Brewers statement of having to carefully craft a system around all the invariants one might want to keep. The exploration of the CAP
space is his single most important point. And in his opinion, the best systems and services are those that have done said deep testing and exploration of CAP. Every system has to consider its own special situation and purpose in order to get the best results.

From my point of view, this is a very valid statement. Sadly there is no one best way of doing things – there rarely is. And as Brewer gives encouragement to make use of the opportunities, the message is also a very positive one. At no point does he talk much about negative results or things that are impossible.

All in all, the publication is an interesting read even for people that have not been involved into the Distributed Information Systems research community for a long time yet since it gives a nice overview of what happened in the past twelve years and what systems currently exist. Starting from CAPs inception, explaining the original formulation and how it was initially received, going to what happened since then, and – as the title says – how the “rules” have changed. All of this is done using many examples, easing the understanding of what Brewer is trying to tell his readers.

Additionally, Brewer builds bridges from existing real world solutions like Google Docs offline mode or the way Facebook serves data to his boiled down model of partition management. In the end, he concludes by giving advice on how future systems should be designed, providing a “take home message” for developers.

So overall, I think Brewer managed to write an article that is both appealing to newbies in the area, while at the same time managing to bring in interesting thoughts and mindsets for more experienced developers, which is a rare thing.

The only problem I had with the publication was summing it up into presentation form and extracting short and concise statements. Even though Brewer manages well to give the reader a feeling of what he tries to explain by giving a lot of examples, it was not quite easy to compile all of it into central bullet points and carry out his message to the listeners of my presentation. This became clear to me again in the discussion after the presentation, as the audience was questioning about the freedom I tried to communicate. Of course, the article was not written with a presentation in mind.

In the end, I think I learned a lot working on both the workshop presentation and report and hope to have also brought some of it to my fellow students. I also thought the subject to be well included into the lecture and other presentations that I listened to; the whole workshop fit quite well together thematically.