There is no free lunch with distributed data white paper

Consistency, availability, and partition-tolerance trade-offs on distributed data access systems

Introduction

Any organization with more than one location must often make important tradeoffs about network communication among its locations. In an ideal world where network bandwidth is infinite, latency is zero, communication cost is free, and links are never down, such tradeoffs would become irrelevant. It would be sensible to centralize all infrastructure and everyone would have cheap, reliable, and easy access to those centralized resources. However, in the real world there are tradeoffs to be made: costs versus what can be purchased in different locations versus physical limits such as the speed of light. People designing networks and distributed applications are familiar with these tradeoffs, and often balance them skillfully in working through a problem.

While many IT experts have great intuition about the design tradeoffs involved with distributed access to business data, it was only recently that the research community codified an important tradeoff in a simple and useful theorem. This theorem relates the tradeoffs that exist among **consistency**, **availability**, and **partition-tolerance (CAP)** for systems that provide distributed access to data. In this terminology, **“consistent”** means that any part of the overall system, if and when it responds to a request for data, provides precisely the correct data. **“Available,”** on the other hand, means that all parts of the overall system are “always up,” and any component will promptly provide an answer to any request. Finally, **“partition-tolerant”** means that the system continues to function in the face of network disruption (or partitions).

The “CAP theorem” says that while it is possible for a system for distributed data access to possess two of the three properties of consistency, availability, and partition-tolerance, it is outright impossible to achieve all three simultaneously. Said another way, when the network goes down (implying the system has no choice but to become partition-tolerant), you must give up either consistency or availability. Either you cannot get to your data (“hey, the network is down!”), or you run the risk of causing data inconsistencies (“hey, someone changed the file I had opened and I lost my work!”). Work by professors at the University of California–Berkeley and MIT means that this conjecture is now a proven theorem. Simply stated, there is no free lunch with distributed data access.

Try as they may…

Certain vendors of file caching systems would like you to believe there is no CAP theorem. Since lack of availability is easy to see, these vendors have typically chosen some kind of availability gain at the expense of consistency or partition-tolerance. The designers then hope that problems are rare enough to be ignored or explained away. Armed with knowledge of the published impossibility results in the scientific literature, it is possible to play detective on these systems and find where the hidden problems are.

File Cache C

File cache system C claims to support disconnected operation. However, system C only attempts to do so for about 1 minute. After that 1-minute window, the remote sites are unable to use the files, which means that the system is no longer available. In addition, system C actually supports only limited availability (reads on open files) during that 1-minute window.

System C can also be operated in a mode where consistency is guaranteed, but then does not support disconnected operation. In addition, system C offers little or no performance gain in that configuration.

File Cache T

File cache system T also claims to support disconnected operation. In the T system, file system locks are held by the server-side unit so as to preserve consistency on behalf of a remote client-side unit. Unfortunately, a partition separating file-using clients from the server for any significant length of time will mean that it is impossible to release such locks at the server. The files are effectively held hostage by the remote client-side unit, and accordingly are not available to any other user of the system. Writes are buffered at the client-side unit in the hope of executing them later, when the partition is repaired. If the client-side unit fails during a partition, writes to files can be permanently lost even when the user believes the data has been successfully saved.

It is possible for an administrator of a T system to break the server-side locks manually. Doing so will not introduce any inconsistency as long as the remote box has only read the locked files. But because of the network partition, there is no way for the administrator to have the system verify this condition before breaking the locks. Instead, the best that can be achieved is to have communication by some other means (such as a manual inspection and a phone call) to verify that breaking locks is acceptable. If the remote files have in fact been written, breaking the locks can cause inconsistency in the data. If the client-side unit has failed, it is impossible to determine what file and lock state it held on behalf of clients.

Likewise, if the server-side file cache fails or reboots during a network partition, then any server-side locks will be relinquished by Common Internet File System (CIFS). This creates a window of opportunity for other clients to access and modify the otherwise locked files. When the network partition heals and the server-side cache recovers, old updates may arrive and overwrite more recent updates to the files, creating inconsistencies and the potential for lost work.

How do HP StorageWorks Enterprise File Services (EFS) WAN Accelerators behave?

HP EFS WAN Accelerators can improve the behavior of distributed applications in situations where network bandwidth and latency are far from ideal. In their usual configuration, they are consistent and partition-tolerant. They will not corrupt data under any circumstances, including unlikely combinations of network and server failures. Because every request is actually sent to the server to generate a response, the usual EFS WAN Accelerator configuration lacks availability in the presence of a network partition (disconnected operation is not possible in this configuration). This is not a failure on the part of the EFS WAN Accelerator; it is an unavoidable tradeoff.

However, there is an additional mode of operation for EFS WAN Accelerators that offers a slightly different tradeoff. With the Proxy File Service enabled, availability can be improved compared to a conventional EFS WAN Accelerator configuration for files. Even in the presence of arbitrary partitions, the “master” copy of a file will continue to be available for all local operations, while the “slave” copies will be available for operations that only read. This arrangement is still consistent: the reads at slave copies during the partition can all be linearized to fit into the instant after the partition, before any writes occurred at the master. The design still represents a tradeoff: at all of the slave copies, write operations are not available.

There is no “free lunch” with distributed data. HP EFS WAN Accelerators offer application acceleration that is always consistent regardless of network and server failures, with the best availability that is possible. The EFS WAN Accelerator’s Proxy File Service configuration allows a choice of higher availability for local file data for applications where that is useful. Competing products claiming higher availability in all situations are necessarily sacrificing consistency, partition-tolerance, or both.

Defining the terms

This section defines terms more carefully, starting by modeling an application as a sequence of transactions (request/response pairs) issued by clients. Each transaction reads or updates part of a single "centralized" shared state. This model is flexible enough to cover the Web, network file systems, and other common examples of client/server systems.

With this model, consistency means that the concurrent operations of the distributed system can be **linearized** into a single (total) ordering that could have occurred in a centralized system. The linearization means that operations at a single client or server have to retain their local ordering exactly as they occurred, but operations that took place at different clients or servers at more or less "the same time" can be interleaved in any arrangement that does not violate any local ordering. This kind of consistency effectively means that the system acts as though it were not distributed. The execution of the distributed application does not introduce any anomalous sequences of events, sequences that could not have occurred in the centralized application.

Availability is simpler to define: it means that every request issued receives a response. Availability corresponds to the end user's perception of whether the system is "up" or "down."

Finally, partition-tolerance means that consistency and/or availability are maintained despite network and server failures where messages may be lost at exactly the "wrong" time or for extended periods of time. A partition-tolerant system may not perform correctly if it is attacked by an active adversary generating false messages, but it will maintain its properties despite any of the more typical kinds of network or server failures, even if those happen in unlikely combinations or sequences.

Now consider the good and bad aspects of systems lacking one of these three attributes (consistency, availability, and partition-tolerance).

- A system lacking consistency (Figure 1) will always be up despite network failures, but will sometimes give incorrect answers. This tradeoff can be acceptable for applications that can tolerate some "slop" in their data, or that have other means of detecting and correcting the errors. This tradeoff is unacceptable in general because the form of the inconsistency is unpredictable. It may be subtle, so that it is widely propagated before it is noticed (if it is ever noticed), like errors introduced into a budget or a contract. Alternately, it may have a large and immediately damaging effect that is irreparable, like causing cash to be dispensed or a weapon to be launched.

What are the anomalous sequences that can occur?

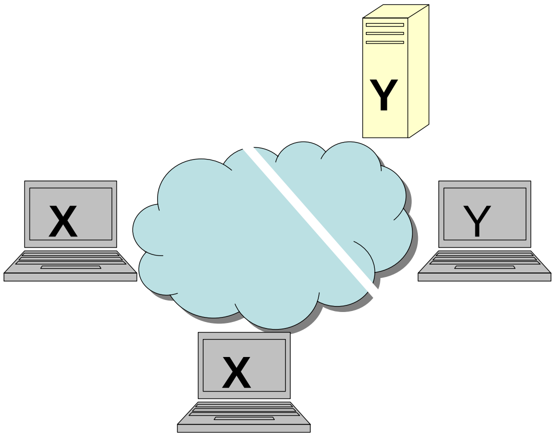
The three problems that arise when interleaving the concurrent actions are: lost update, dirty read, and unrepeatable read.

A lost update happens when two transactions are each reading and updating the same state—whichever writes later "wins" and the previous update value is lost. In a sequential execution, one transaction would have written its state first, and the second would have read that transaction's results as a basis for its update.

A dirty read happens when a second transaction reads a state that is a temporary or intermediate value written by a first transaction. The second transaction's results depend on an incorrect value that was not intended to be exposed by the first transaction, and could not have been seen in a sequential execution.

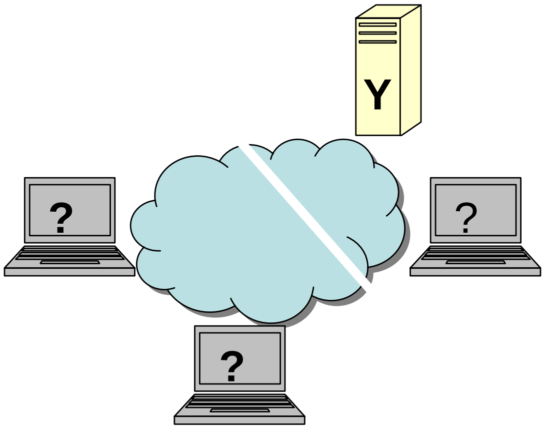
An unrepeatable read happens when a transaction reads the same state twice—expecting to read the same value but instead receiving the results of some other transaction's update the second time.

Figure 1. A system lacking consistency



- A system lacking availability (Figure 2) will tolerate network failures without corruption or anomalies, but will be "down" whenever network failures make it impossible to perform transactions while staying consistent. These situations can be frustrating for users and administrators because the relevant data may appear to be locally available. It may look as though operating on that local data should be fine, but in the context of potentially arbitrary network failures, it is sometimes impossible to be sure that it will actually be safe.

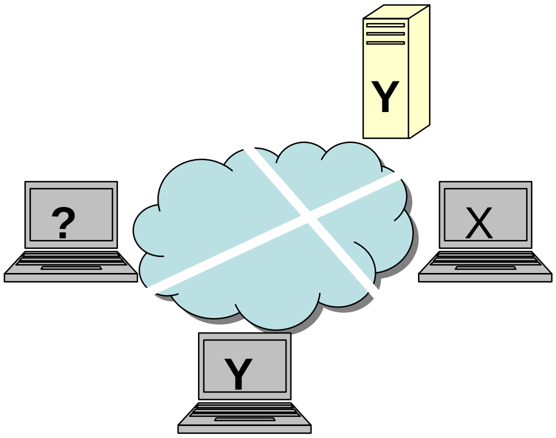
Figure 2. A system lacking availability



- A system lacking partition-tolerance (Figure 3) will appear to be both consistent and available as long as its network is well-behaved. Although it is possible to improve the probability of having a well-behaved network by using diverse redundant elements, it is not possible to ensure that partitions never occur.

  If a system lacks partition-tolerance, then things become uncertain in the case where a partition does occur: some designs will fail by being inconsistent, others will fail by being unavailable, and still others will fail in both ways at once.

Figure 3. A system lacking partition-tolerance



Is it possible for a system to have all three attributes at once? This question turns out to have been of interest to mathematicians and computer scientists for some time.

What research indicates

The theoretical analysis of the tradeoff between consistency, availability, and partition-tolerance takes the form of impossibility proofs. An impossibility proof sets up a careful model of a system and some interesting condition, and then proceeds to demonstrate that the interesting condition is impossible within the system as modeled. Often these proofs proceed by a contradiction argument, assuming the interesting condition is true and then exposing some contradiction between that assumption and some other necessary aspect of the modeled system.

The first relevant impossibility result was published in 1986[1]. The authors of that paper made four simplifying assumptions:

- The data items are all replicated at every site.
- The mix of transactions is similar at each site.
- There are no "blind writes," transactions that update data items without reading them.
- There is only a single partition, dividing the sites into a majority side and a minority side.

[1] Brian A. Coan, Brian M. Oki, and Elliot K. Kolodner. "Limitations on Database Availability when Networks Partition." *Proceedings of the Fifth Annual ACM Symposium on Principles of Distributed Computing* (1986), pp. 187–94.

Comparing these assumptions to the real world, you can see that the assumptions of uniform data and transactions are a sensible approximation of what can be achieved without application-specific tuning. Non-uniform distributions might sometimes do better and sometimes do worse than a uniform distribution when application characteristics are unknown, but a uniform distribution of data and transactions provides a sound baseline for analysis.

The assumption of a single majority/minority partition is the best case when a partition occurs. Other configurations such as an even split (neither side a majority) or multi-way split (additional minorities) do not offer any improvement in availability.

That is, availability is the fraction of total transactions presented during the partition that successfully complete. A completely available system achieves availability of 1. With the following definitions:

u_{maj} = the fraction of update transactions in the majority partition

r_{maj} = the fraction of read-only transactions in the majority partition

r_{min} = the fraction of read-only transactions in the minority partition

They proved that for any online replicated-data management technique that maintains consistency,

$$Availabili\,ty \leq u_{maj} + r_{maj} + r_{min}$$

One way of understanding this result is that the best implementation under these assumptions can only complete all transactions on the majority side, plus the read-only transactions on the minority side. Another way of thinking about this is that for a system to achieve 100% availability, it must be designed to support only read-only transactions on the minority side during a partition.

A more recent result was published in 2002.[2] This later work made no assumption regarding uniformity of replication or transactions, and is accordingly a stronger confirmation of the tradeoff.

The authors of the second paper proved impossibility results in two different network models: **asynchronous** (no clocks at all) and **partially synchronous** (local clocks). In a partially synchronous network, each site has a local clock and all clocks increment at the same rate, but the clocks are not synchronized to display the same value at the same instant. The partially synchronous network corresponds to a likely best-case scenario in the presence of partitions: clocks are uniform in behavior but it is not possible to rely on their synchronization.

Roughly speaking, it is harder to do the right thing in an asynchronous network than in a partially synchronous network. Accordingly, the results of most interest are what is **impossible** in the "easier" partially synchronous network, and what is **possible** in the "harder" asynchronous network.

In the partially synchronous network, the authors proved that it is impossible to implement a read/write data object that guarantees both availability and consistency in all executions, even those in which messages are lost. This is a proof that all three properties (consistency, availability, and partition-tolerance) cannot be achieved simultaneously in a realistic network model. However, even in the asynchronous network model, where additional desirable conditions are provably impossible, they showed that it is possible to achieve any two-of-three combination.

[2] Seth Gilbert and Nancy Lynch. "Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services." *Sigact News*, 33(2), June 2002.

Putting it all together

There is no “free lunch” with distributed data. The “CAP theorem” research described shows that one of the three (consistency, availability, and partition-tolerance) has to be traded away.

HP EFS WAN Accelerators offer application acceleration that is always consistent regardless of network and server failures, with the best availability that is possible.

The EFS WAN Accelerator’s Proxy File Service configuration allows a choice of higher availability for local file data for applications where that is useful. Competing products claiming higher availability in all situations are necessarily sacrificing consistency, partition-tolerance, or both.

For more information

For more information on HP StorageWorks EFS WAN Accelerators, visit:

http://www.hp.com/go/efs

For additional HP StorageWorks EFS Accelerator white papers, visit:

http://h18006.www1.hp.com/storage/efswhitepapers.html

5983-2544EN, 05/2005