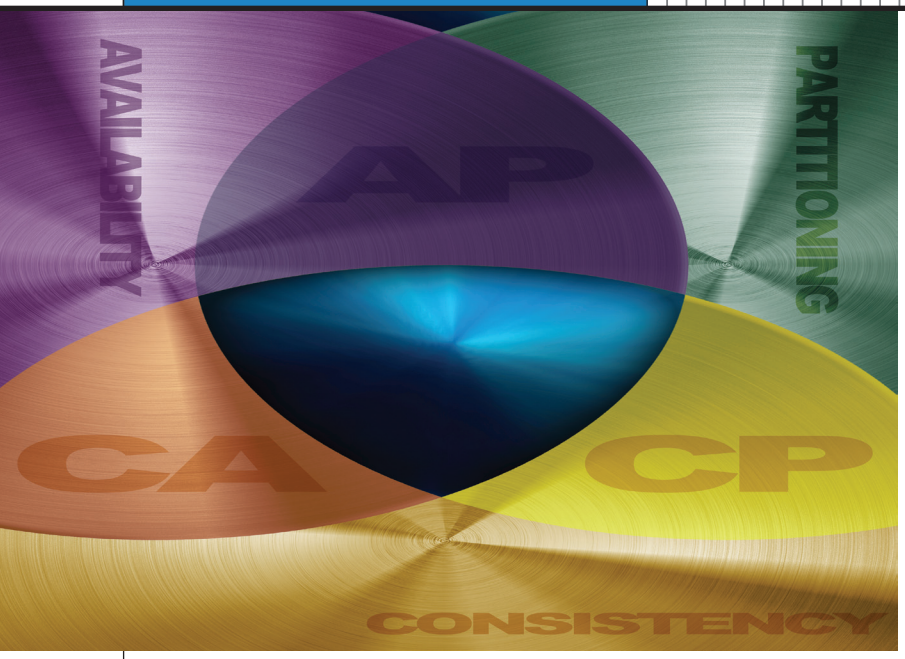

The CAP Theorem's Growing Impact

Simon S.Y. Shim, *San Jose State University*

The computing community has been developing innovative solutions to meet the formidable challenge of handling the exponential growth in data generated by the Web.

For a long time, commercial relational database management systems with ACID (atomicity, consistency, isolation, and durability) properties from vendors such as Oracle, IBM, Sybase, and Microsoft have been the default home for computational data. However, with the phenomenal growth of Web-generated data—which Vint Cerf referred to as an "information avalanche"—this conventional way of storing data has encountered a formidable challenge.

Because the traditional way of handling petabytes of data with a relational database in the back end does not scale well, managing this phenomenon referred to as the *big data challenge* has become problematic. Highly scalable database solutions are needed to meet the demand of handling this data explosion. With the abundance of inexpensive commodity servers, public and private clouds can store and process big data effectively by scaling horizontally into distributed systems.

To manage the exponentially growing data traffic, Internet companies such as Google, Amazon, Yahoo, Facebook, and Twitter have developed alternative solutions that store data in what have come to be known as NoSQL databases. Carlo Strozzi coined this term in 1990 to refer to databases that do not use SQL as their query language, and Eric Evans later used it to refer to nonrelational and distributed databases. Internet companies contributed their NoSQL databases as open source projects, which later became popular for storing large data.

In general, NoSQL databases support flexible schema, scale horizontally, and, more interestingly, do not support ACID properties. They store and replicate data in distributed systems, often across datacenters, to achieve scalability and reliability.

To tolerate network partitions and limit write latency, these systems relax the consistency requirement so that data updates are performed asynchronously, while potential data conflicts can be resolved at data reads. Hence, the system might return inconsistent values from distributed data stores depending on where it read the data. Readers must resolve the potential data inconsistency.

With the advances in datacenter network infrastructures, network failures are rare, and this tradeoff between network partitions and data consistency is less relevant within a single datacenter. However, this remains a significant challenge for cloud providers who must maintain multiple datacenters in geographically separated regions. The computing community has been developing innovative solutions to address this issue.

IN THIS ISSUE

In late 2000, Eric Brewer gave a talk on his famous CAP conjecture at the Principles of Distributed Computing Con-

0018-9162/12/$31.00 © 2012 IEEE Published by the IEEE Computer Society

ference. According to the CAP theorem, it is only possible to simultaneously provide any two of the three following properties in distributed Web-based applications: *consistency* (C), *availability* (A), and *partition* tolerance (P). Later, Seth Gilbert and Nancy A. Lynch proved the conjecture under certain circumstances.

In "CAP Twelve Years Later: How the 'Rules' Have Changed," Eric Brewer explains that when it is necessary to choose between C and A during network partitioning, the designer often chooses A. In "Perspectives on the CAP Theorem," Seth Gilbert and Nancy A. Lynch review the theorem and discuss its practical implications.

Cloud providers have broadened the interpretation of the CAP theorem in the sense that they consider a system as not available if the response time exceeds the latency limit. Thus, a slow network link is considered partitioned. To obtain extreme scalability and performance without node-to-node coordination or synchronization, developers relax consistency to make the system "available" even when there is no real network failure. As Werner Vogels, chief technology officer and vice president of Amazon, puts it, "We do not want to relax consistency. Reality, however, forces us to." But many applications need consistency.

In "Consistency Tradeoffs in Modern Distributed Database System Design," Daniel J. Abadi clarifies that a distributed system often cannot support synchronous replication because many applications require low latency. Thus, consistency is sacrificed even when there is no network partition. To better understand the tradeoffs of the CAP theorem, Abadi suggests rewriting CAP, making the latency versus consistency tradeoff explicit.

Building a highly available system by choosing availability over consistency is bound to increase the complexity of distributed systems. Data inconsistency imposes a dimension of design complexity in application development. Programmers need to know when to use fast/inconsistent accesses versus slow/consistent accesses to secure both high performance and correctness. Moreover, they might need to define the conflict resolution rules that meet the application's needs.

Developers have studied a few weak consistency models, which can be understood through the tradeoff between design complexity and availability—and thus, performance. Eventual consistency chooses availability, thus allowing updates to any closest replica. The system eventually propagates updates over time. However, this eventual consistency might not be able to guarantee the same order of updates.

Application-specific rules are needed to resolve data conflicts. An example of this is Amazon's shopping cart built using Dynamo, Amazon's eventually consistent key-value store. This eventual consistency concept was first used in the early 1990s in the Coda, Bayou, and Ficus distributed systems.

The developers of Yahoo's PNUTS chose to provide stronger consistency at the expense of availability. PNUTS provides timeline consistency on a per-tuple basis, which guarantees that each tuple will undergo the same sequence of transformations at each storage replica. PNUTS employs a per-tuple master node that orders all updates made to the tuple, after which the system disseminates them asynchronously to the storage replicas. Conflict resolution becomes simpler, as storage replicas now see updates in the same order.

However, coordinating all updates through a master may have obvious performance and availability implications. PNUTS alleviates these issues by automatically migrating the master to be close to the writers. As Raghu Ramakrishnan points out in "CAP and Cloud Data Management," this makes the practical impact on performance and availability insignificant for Yahoo's applications because of localized user access patterns.

Finally, in "Overcoming CAP with Consistent Soft-State Replication," Kenneth P. Birman and his coauthors advocate for even stronger consistency inside the datacenter, where partitions are rare. They show that in this setting, it is possible to achieve low latency and scalability without sacrificing consistency.

With data generation guaranteed to grow rather than shrink, the articles included in this special issue demonstrate that, 12 years after Eric Brewer first proposed it, the CAP theorem continues to play a crucial role in understanding and optimizing distributed systems.

Simon S.Y. Shim *is a professor in the Computer Engineering Department at San Jose State University. Contact him at simon.shim@sjsu.edu.*

INTERESTED IN LEARNING MORE ABOUT CAP?

In "NoSQL and Mongo DB with Dwight Merriman," Software Engineering Radio features an interview with Merriman about the emerging NoSQL movement, the three types of nonrelational data stores, Brewer's CAP theorem, the weaker consistency guarantees that can be made in a distributed database, document-oriented data stores, the data storage needs of modern Web applications, and the open source MongoDB: www.se-radio.net/2010/07/episode-165-nosql-and-mongodb-with-dwight-merriman.