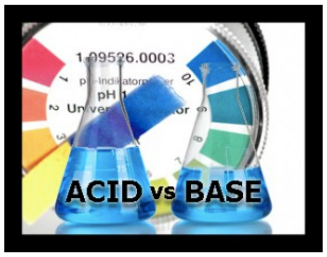
ACID vs. BASE: The Shifting pH of Database Transaction Processing

By Charles Roe on March 1, 2012



In Chemistry, pH measures the relative basicity and acidity of an aqueous (solvent in water) solution. The pH scale extends from 0 (highly acidic substances such as battery acid) to 14 (highly alkaline substances like lie); pure water at 77° F (25° C) has a pH of 7 and is neutral. Data engineers have cleverly borrowed acid vs base from chemists and created acronyms that while not exact in their meanings, are still apt representations of what is happening within a given database system when discussing the reliability of transaction processing.

To extend this useful metaphor further, let's first look at the idea of pH. Søren Peder Lauritz Sørensen first developed the concept while working at the Carlsberg Laboratory in 1909, so the idea is not something new. The (H) refers to hydrogen atoms. There is considerable disagreement whether the (p) comes from the French *puissance*, German *potenz*, or Latin *pondus* or *potentia*; yet, they all mean potential or power. Thus, pH is the power/potential energy of hydrogen in a given solution, or the *activity* of the hydrogen atoms in that solution: an acidic solution has more activity, an alkaline solution has less. In database transaction processing, the hydrogen atoms are the data flowing through the system; the (p) are the properties that guarantee those transactions are processing reliably.

What is ACID?

Ask any data professional and they could probably explain the ACID (Atomicity, Consistency, Isolation, and Durability) acronym quite well. The concept has been around for decades and until recently was the primary benchmark that all databases strive to achieve – without the ACID requirement in place within a given system, reliability was suspect.

Jim Grey first conceived of the idea in the 1970s and subsequently published the groundbreaking paper "The Transaction Concept: Virtues and Limitations" in June 1981. The paper only discussed the terms Atomicity, Consistency and Durability; Isolation was added later. It focused on the transaction level and defined the transaction as a contract or any number of "transformations of a system state" that had to have CAD as its inherent properties or "system consistency constraints." Bruce Lindsay et al. wrote the paper "Notes on Distributed Databases" in 1979 that built upon Grey's work, laid down the fundamentals for achieving consistency, and the primary standards for database replication. In 1983, Andreas Reuter and Theo Härder published the paper "Principles of Transaction-Oriented Database Recovery" and officially coined the term ACID, which some two decades later has come to mean:

- **Atomicity:** Either the task (or all tasks) within a transaction are performed or none of them are. This is the all-or-none principle. If one element of a transaction fails the entire transaction fails.
- **Consistency:** The transaction must meet all protocols or rules defined by the system at all times. The transaction does not violate those protocols and the database must remain in a consistent state at the beginning and end of a transaction; there are never any half-completed transactions.
- **Isolation:** No transaction has access to any other transaction that is in an intermediate or unfinished state. Thus, each transaction is independent unto itself. This is required for both performance and consistency of transactions within a database.
- **Durability:** Once the transaction is complete, it will persist as complete and cannot be undone; it will survive system failure, power loss and other types of system breakdowns.

There are of course many facets to those definitions and within the actual ACID requirement of each particular database, but overall in the RDBMS world, ACID is overlord and without ACID reliability is uncertain. The pH of ACID is low, roughly similar to battery acid (0) or maybe vinegar (2), the data and its constraints are exceedingly active. Therefore, at any given microsecond in a database that uses ACID as its system constraint all the data (hydrogen atoms) are undergoing constant checks to make sure they fulfill those constraints. Such requirements had

worked quite well for many years in the smaller, horizontally scalable, schema-driven, normalized, relational world of the Pre-Social Networking bygone age. Such past truisms are longer the case; Unstructured Data, Big Data , non-relational data structures, distributed computing systems and eventual consistency are now becoming more commonplace; new requirements mean new acronyms and a new pH.

CAP Theorem

In 2000, Eric Brewer presented his keynote speech at the ACM Symposium on the Principles of Distributed Computing and CAP Theorem was born. Most people in the outside world have never heard of such a theorem nor do they care; they just want their PCs to work, the Internet to work, social media to work, all with consistent availability to their files. CAP Theorem, also known as Brewer's Theorem, was later revised and altered through the work of Seth Gilbert and Nancy Lynch of MIT in 2002, plus many others since. The central tenet of the theorem states that there are three essential system requirements necessary for the successful design, implementation and deployment of applications in distributed computing systems. They are **Consistency**, **Availability** and **Partition Tolerance** – or **CAP**:

- **Consistency** refers to whether a system operates fully or not. Does the system reliably follow the established rules within its programming according to those defined rules? Do all nodes within a cluster see all the data they are supposed to? This is the same idea presented in ACID.
- **Availability** means just as it sounds. Is the given service or system available when requested? Does each request get a response outside of failure or success?
- **Partition Tolerance** represents the fact that a given system continues to operate even under circumstances of data loss or system failure. A single node failure should not cause the entire system to collapse.

Those are only simple definitions of the three aspects of CAP Theorem ; there are numerous papers available that discuss the many interpretations, analyses and complex issues involved in the actual real-world application of the theorem.

The primary reason for presenting the theorem is to make the point that in the majority of instances, a distributed system can only guarantee two of the features, not all three. To ignore such a decision could have catastrophic results that include the possibility of all three elements falling apart simultaneously. The constraints of CAP Theorem on database reliability were monumental for new large-scale, distributed, non-relational systems: they often need Availability and Partition Tolerance, so Consistency suffers and ACID collapses. "Run for the hills" is an apt phrase.

BASE Introduces Itself and Takes a Bow

Luckily for the world of distributed computing systems, their engineers are clever. How do the vast data systems of the world such as Google's BigTable and Amazon's Dynamo and Facebook's Cassandra (to name only three of many) deal with a loss of consistency and still maintain system reliability? The answer, while certainly not simple, was actually a matter of chemistry or pH: **BASE** (**B**asically **A**vailable, **S**oft state, **E**ventual consistency). In a system where BASE is the prime requirement for reliability, the activity/potential (p) of the data (H) changes; it *essentially* slows down. On the pH scale, a BASE system is closer to soapy water (12) or maybe the Great Salt Lake (10). Such a statement is not claiming that billions of transactions are not happening rapidly, they still are, but it is the constraints on those transactions that have changed; those constraints are happening at different times with different rules. In an ACID system, the data fizzes and bubbles and is perpetually active; in a BASE system, the bubbles are still there much like bath water, popping, gurgling, and spinning, but not with the same vigor required from ACID. Here is why:

- **Basically Available:** This constraint states that the system does guarantee the availability of the data as regards CAP Theorem; there will be a response to any request. But, that response could still be 'failure' to obtain the requested data or the data may be in an inconsistent or changing state, much like waiting for a check to clear in your bank account.
- **Soft state:** The state of the system could change over time, so even during times without input there may be changes going on due to 'eventual consistency,' thus the state of the system is always 'soft.'
- **Eventual consistency:** The system will *eventually* become consistent once it stops receiving input. The data will propagate to everywhere it should sooner or later, but the system will continue to receive input and is not checking the consistency of every transaction before it moves onto the next one. Werner Vogel's article "Eventually Consistent – Revisited" covers this topic is much greater detail.

Conclusion – Moving Forward

The new pH of database transaction processing has allowed for more efficient horizontal scaling at cost effective levels; checking the consistency of every single transaction at every moment of every change adds gargantuan costs to a system that has literally trillions of transactions occurring. The computing requirements are even more astronomical. Eventual consistency gave organizations such as Yahoo! and Google and Twitter and Amazon, plus thousands (if not millions) more the ability to interact with customers across the globe, continuously, with the necessary availability and partition tolerance, while keeping their costs down, their systems up, and their customers happy. Of course they would all like to have complete consistency all the time, but as Dan Pritchett discusses in his article "BASE: An Acid Alternative," there has to be tradeoffs, and eventual consistency allowed for the effective development of systems that could deal with the exponential increase of data due to social networking, cloud computing and other Big Data projects.

This article has been so popular, we have expanded upon the content into a White Paper:

The Question of Database Transaction Processing: An ACID, BASE, NoSQL Primer

Further Reading

Bartels, D. Taking NoSQL 1.0 on a Journey into the Enterprise [1-part video]. Retrieved from http://www.dataversity.net/archives/6600 .

Bloor, R. The Coming Database Revolution. Retrieved from http://www.dataversity.net/archives/5638 .

DeCandia, G., Hastorun, D., Jampani, M., Kakulapati, G., Lakshman, A., Pilchin, A., Sivasubramanian, S., Vosshall, P., Vogels, W. Dynamo: Amazon's Highly Available Key-value Store. Retrieved from http://s3.amazonaws.com/AllThingsDistributed/sosp/amazon-dynamo-sosp2007.pdf .

Ingenthron, M. How AOL Advertising Uses NoSQL to Make Millions of Smart Targeting Decisions Every Hour [1-part video]. Retrieved from http://www.dataversity.net/archives/6786 .

Kerner, S.M. Inside Facebook's Open Source Infrastructure. Retrieved from http://www.developer.com/open/article.php/3894566/Inside-Facebooks-Open-Source-Infrastructure.htm .

Hsieh, J. Cloud Deployment of Hadoop and HBase [4-part video]. Retrieved from http://www.dataversity.net/archives/6530 .

Meir-Huber, M. NoSQL – The Trend for Databases in the Cloud? Retrieved from http://soa.sys-con.com/node/1615716 .

Schireson, M. Re-inventing the Database: What to Keep and What to Throw Away [1-part video]. Retrieved from http://www.dataversity.net/archives/6781 .

Thomas, G.J. Big Data on the Micro Scale. Retrieved from http://www.dataversity.net/archives/7287 .

Thomas, G.J. Government Data in the Cloud: Things to Consider. Retrieved from http://www.dataversity.net/archives/5864 .

Upton, T. A Conversation with Bruce Lindsay. Retrieved from http://queue.acm.org/detail.cfm?id=1036486 .

2 Comments DATAVERSITY 1 Login

Recommend 2 Tweet Share Sort by Best

Join the discussion…

LOG IN WITH OR SIGN UP WITH DISQUS ?

Name

SacTiw · 2 years ago

>>> This is the same idea presented in ACID.

Now this part confuses me, for what I have read so far is 'C' in CAP is not similar to to 'C' in ACID i.e. they are different. Can you clarify further?

Reply · Share ›

Binh Thanh Nguyen · 5 years ago

Thanks, very nice comparison

Reply · Share ›

ALSO ON **DATAVERSITY**

Top Programming Languages for Data Science and Machine Learning
2 comments • 5 months ago
Antonello Lobianco — I think it is clear here "beside R and Python"

So You Want to be a Data Governance Specialist?
2 comments • 3 months ago
rachael fay — This is interesting!!! I didn't think a degree or 3-4 years work experience in the data world can you make you a data governance

Where is Your Organization on the Digital Transformation Journey?
1 comment • 2 months ago
Ahmed Alnemari — Very insightfulThanks

The Challenge of Scaling Transactional Databases
1 comment • 4 months ago
Joel Mamedov — Large data, scalable transaction. Transactions (if you design properly) should not deal with large data set.

Subscribe   Add Disqus to your site   Disqus' Privacy Policy   DISQUS

DATAVERSITY   DATAVERSITY.net   TDAN.com   DMRADIO.biz

Conferences
Enterprise Data World
Data Architecture Summit
DG Vision

Online Conferences
Enterprise Data Governance Online
Data Architecture Online
Enterprise Analytics Online

DATAVERSITY Resources
DATAVERSITY Community
White Papers
What is…?
Concept and Object Modeling Notation (COMN)

DGIQ Data Governance & Information Quality CONFERENCE DGIQCONFERENCE.COM

DATA ARCHITECTURE SUMMIT DATAARCHITECTURESUMMIT.COM

ENTERPRISE DATA WORLD ENTERPRISEDATAWORLD.COM

Company Information
About Us
Advertise With Us
Contact Us
Press Room

Newsletters
DATAVERSITY Weekly
DATAVERSITY Community Weekly
TDAN.com

DATAVERSITY Education
Data Conferences
Trade Journal
Online Training
Upcoming Live Webinars
Books

Privacy Policy Terms of Service Cookies Settings