Consistently Eventual

**FOR MANY
DATA ITEMS,
THE WORK NEVER
SETTLES ON
A VALUE.**

PAT HELLAND

In recent years, there has been a lot of excitement over eventual consistency.⁶ Heck, I get pretty excited about it! is an aspect of some data that says its underlying value is unknown until work on that item settles down. It turns out that, in many cases, there are data items for which the work *never* settles down. In addition to being *eventually consistent*, many data items remain *consistently eventual*!

WHAT IS EVENTUAL CONSISTENCY?

Eventual consistency occurs when the value for something is replicated in more than one place, and there's a protocol for these replicas converging. Changes to one or more of the replicas can be done independently, and they will propagate and converge.

Eventual consistency

When we all know the same stuff, we'll have the same result.

In this article, I don't want to talk about how eventual consistency can be accomplished but more about what it looks like when it's used. Many fun papers have been written about eventual consistency. One of my favorites is "Eventual Consistency Today: Limitations, Extensions, and Beyond," by Peter Bailis and Ali Ghodsi.¹

As already mentioned, eventual consistency is typically

*It's Not Your Grandmother's Database Anymore

used to describe the behavior of a data item that is replicated over decoupled systems. When updates happen to disconnected replicas separately, how do they behave when they reconnect and share their state?

Still, eventual consistency typically refers to the *behavior of a single replicated object*. It doesn't usually speak to transactions and what they mean with eventually consistent objects.

DATA FLOATING LOOSE IN THE MEAN, CRUEL WORLD

Data in a relational database behaves differently from data kept outside of one. When nonrelational data is unlocked, it gets captured as a message, file, key value, or something else grouped as a lump. These lumps (or objects or values or entities...) have an identity and a version.²

Eventual consistency arises when an object with multiple replicas, *each with the same identity*, somehow coalesces to a common value— *even when the different replicas are updated independently*. This inherently means the version(s) of the object are not linearly assigned. It no longer makes sense to talk about a strict ordering of the changes. You must be prepared to capture the version of the object in a fashion that represents independent changes coming together. An excellent versioning mechanism is the vector clock.

It is important to recognize that the entire eventual consistency discussion must necessarily work in a world with objects, identities, and versions. It's not really a classic database thing.

One problem with having replicas is that you really never know when one of your evil twins will pop back into existence.

WAIT... WHAT'S A TRANSACTION MEAN?

“Last Writer Wins” is a form of eventual consistency. Consider a system that captures the wall-clock time from the local system whenever it updates a replica. When everyone has heard all the updates, the one written with the latest wall-clock time is kept everywhere. This is challenging for transactional updates. Sometimes, a change within a transaction has the latest time and is kept. Sometimes, the transactional change is stomped out by a later update. This makes atomicity a challenge.

A change can also be captured as a commutative operation.⁵ This is evident in banking when a debit or credit is applied to your account and these operations can be reordered or commuted.

When you write a check on your joint checking account and your spouse writes a check at the same time, hopefully they will both clear. Given enough money in the bank, it doesn't matter whose check clears first. A transaction can deposit money via a check into one account drawn on another. The check will usually clear, making a valid transaction. Sometimes, a bounced check will form another transaction to compensate for the first transaction with the bum check.

When dealing with all of these issues, the best you can hope for is a probabilistic success combined with eventual compensation.³ While we strive for perfect transactional work in banking, we end up compensating when stuff goes wrong. Unlike some other areas of human endeavor, for the most part we can compensate for financial errors.

WHEN IS EVENTUAL? IS IT NOW?

One problem with having replicas is that you really never know when one of your evil twins will pop back into existence. Sometimes, algorithms codify that a replica is *persona non grata* after a certain period of time. Sometimes, you overlook that and a zombie replica will come back when least expected.

My wife and I have a checking account that is perennially in a state of debits and credits. When we both use it, no one really knows how much money is flying and floating.

Our personal checking account is *consistently eventual*.

The only way to figure out the balance is to stop using the account for a while.

There's also the pesky problem of the check written to someone who doesn't deposit it in a reasonable amount of time. Perhaps, it was left in a wallet and deposited a year later. If the check isn't deposited for months, do you put a stop order on it or just wait and assume it's not coming through? The balance in your checking account is annoyingly eventual!

SNAPPING UNCERTAINTY INTO "WHAT WE KNOW SO FAR"

Our bank sends us monthly statements. They represent the debits and credits to our account that cleared a strongly consistent location as of a deadline. That strongly consistent location is the bank's centralized computer system. The debits and credits that have arrived at the clearinghouse by the monthly deadline get scooped up into the account's statement.

Quarterly reports for public companies take a similar but different approach. At midnight of the last day of the quarter, new business and new expenses start being allotted to the next quarter. The company begins gathering and organizing all the income and expenses from the now-closed quarter. Then all records of what was spent and earned are swept into a big *mélange* that results in a public quarterly profit and loss report. This usually happens within 40 days or so after the quarter closes.

Some transactions during the quarter, however, may not be sent to the accountants in a timely fashion. One contributing factor may be the employment of software engineers who are notoriously bad at the punctual submission of expense reports. So, the results published for the quarter are approximately correct but not perfect.

After the quarterly report, corrections will dribble into the accountants. They will either categorize them as minor and issue a slight correction to the numbers for the previous quarter or issue a restatement. For a public company, a restatement showing a noticeable difference from the published report is embarrassing and rarely happens. Minor corrections are common.

You can't really know what happened until you've heard everything.

The longer you wait, the more you hear and the more accurate your opinion. Eventually, you give up waiting for new information and declare your opinion of what happened *a few months ago* is accurate enough.

In the bank account statement, the definition of certainty is provided at the bank's centralized computer

system at end-of-day when the month closes. Uncertainty *from the perspective of the bank* is eliminated. In the corporate quarterly reports, uncertainty is gauged by how much of the underlying truth of the business filters its way back to the accountants. The quarterly report is not definitive, just pretty darned close—at least usually.

TRUST, TIMEOUTS, AND ESCALATION

Working across trust boundaries is always eventual. Because you may not trust another entity, you're not going to do a distributed two-phase commit⁴ and lock up your database waiting for that other company. Instead, you have a workflow in which partial trust is used to get your cooperative business done. Throughout this process, there are long windows in which you are just waiting and waiting... still more examples of being consistently eventual.

This eventual nature of uncertainty continues through the steps of the workflow. You agree to reserve 200 widgets from your inventory on the receipt of a deposit from your purchaser. If you hear nothing back to consummate the purchase of the widgets, you are stuck. While your disappointment is somewhat tempered by the deposit you keep, it's not enough to pay for the widgets. Darn! You lost out selling them to the other customer!

Working across trust boundaries, cooperative work functions using timeouts and escalation. If you don't call to cancel your room reservation at a hotel with 72 hours' advance notice, you're stuck with the first night's room charge. The hotel, however, is likely stuck with the remaining six days of an empty room from your one-week reservation.

Indeed, the hotel has an ongoing parade of eventually resolved room sales. By the time it knows what's happening on Tuesday, the confusion of Wednesday's occupancy is

about to get resolved. Again, payments to the hotel are consistently eventual.

Related articles

➡ Don't Settle for Eventual Consistency
Wyatt Lloyd, et al.
Stronger properties for low-latency
geo-replicated storage
<https://queue.acm.org/detail.cfm?id=2610533>

➡ Eventually Consistent:
Not What You Were Expecting?
Wojciech Golab, et al.
Methods of quantifying consistency
(or lack thereof) in eventually consistent
storage systems
<https://queue.acm.org/detail.cfm?id=2582994>

➡ Scalable SQL
Michael Rys, Microsoft Corp.
How do large-scale sites and
applications remain SQL-based?
<https://queue.acm.org/detail.cfm?id=1971597>

CONCLUSION

Applications are no longer islands. Not only do they frequently run distributed and replicated over many cloud-based computers, but they also run over many hand-held computers. This makes it challenging to talk about a single truth at a single place or time. In addition, most modern applications interact with *other* applications. These interactions settle out to impact understanding. Over time, a shared opinion emerges just as new interactions add increasing uncertainty. Many business,

personal, and computational “facts” are, in fact, uncertain. As some changes settle, others meander from place to place.

With all the regular, irregular, and uncleared checks, my understanding of our personal joint checking account is a bit hazy. While I try to convince myself I'll someday understand it, I've reconciled myself that it's really *consistently eventual*.

References

1. Bailis, P., Ghodsi, A., 2013. Eventual consistency today: limitations, extensions, and beyond. *acmqueue* 11(3); <https://queue.acm.org/detail.cfm?id=2462076>.
2. Helland, P. 2005. Data on the outside versus data on the inside. *Proceedings of the Conference on Innovative Database Research*.
3. Helland, P., Campbell, D. 2009. Building on quicksand. *Proceedings of the Conference on Innovative Database Research*.
4. Mohan, C., Lindsay, B. 1985. Efficient commit protocols for the tree of processes model of distributed transactions *ACM SIGOPS Operating Systems Review* 19(2): 40-52; <https://dl.acm.org/citation.cfm?id=850772>.
5. Shapiro, M., Preguiça, N., Baquero, C., Zawirski, M. 2011. Conflict-free replicated data types. *Proceedings of the 13th International Conference on Stabilization, Safety, and the Security of Distributed Systems*. Springer-Verlag, Berlin 386–400.
6. Vogels, W. 2009. Eventually consistent. *Communications of the ACM* 52(1): 40-44; <https://dl.acm.org/citation.cfm?id=1435432>.

Pat Helland has been implementing transaction systems, databases, application platforms, distributed systems, fault-tolerant systems, and messaging systems since 1978. For recreation, he occasionally writes technical papers. He currently works at Salesforce.

Copyright © 2018 held by owner/author. Publication rights licensed to ACM.