Blockchain, Throughput, and Big Data

Trent McConaghy

Bitcoin Startups Berlin
Oct 28, 2014

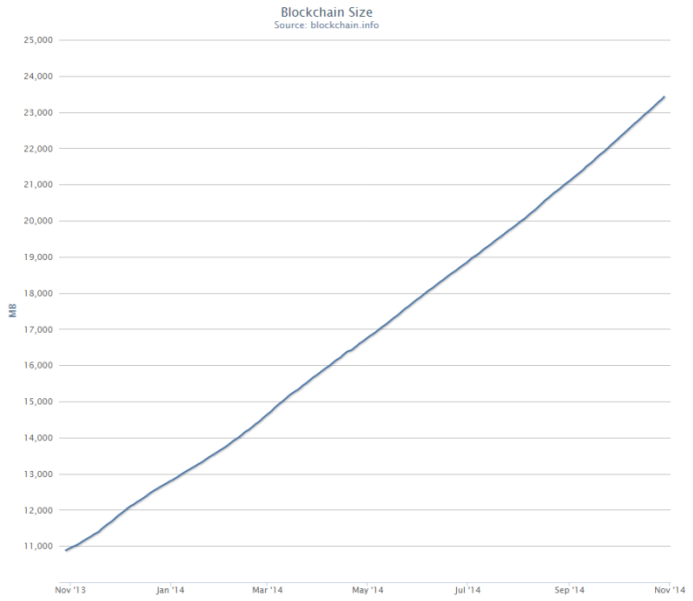
Outline

- Throughput numbers
- Big data
- Consensus algorithms
- ACID
- Blockchain ←→ Big data?

Throughput numbers

- Bitcoin – typical 1 transactions per second
  - Blockchain size is 25Gb. >10Gb / yr.
  - Takes ≈1d to download

25 Gb



11 Gb

Nov’13 Nov’14

[blockchain.info, retrieved Oct 29, 2014]

Throughput numbers

- Bitcoin – 1 tps (observed)
- Bitcoin – 7 tps (theoretical max due to block size limit)
- VISA – 2000 tps
- Twitter – 5000 tps
- Twitter – 150,000+ tps at peak

[https://en.bitcoin.it/wiki/Scalability]

https://blog.twitter.com/2013/new-tweets-per-second-record-and-how

Throughput numbers

What ifs on Bitcoin, all else equal:

- 2000 tps → 10Gb*2000/7 = 1.42 Pb/yr
  = 3.9 Gb/day (grows faster than you can download!)
- 150,000 tps →214 Pb/yr

You might say “we only need unspent outputs”. But there are many use cases where we want to see all past transactions (blockchain apps, transparency, auditing, .. – almost anything beyond simple payment.)

Throughput numbers

What ifs on Bitcoin, all else equal:

- 2000 tps → 10Gb*2000/7 = 1.42 Pb/yr
  = 3.9 Gb/day (grows faster than you can download!)
- 150,000 tps →214 Pb/yr

Q: Why not just focus on unspent outputs?

A: There are many use cases where we want to see all past transactions. Auditing, compliance, ownership history, other Blockchain apps -- maybe most things beyond a simple payment?)

Scaling up blockchain?

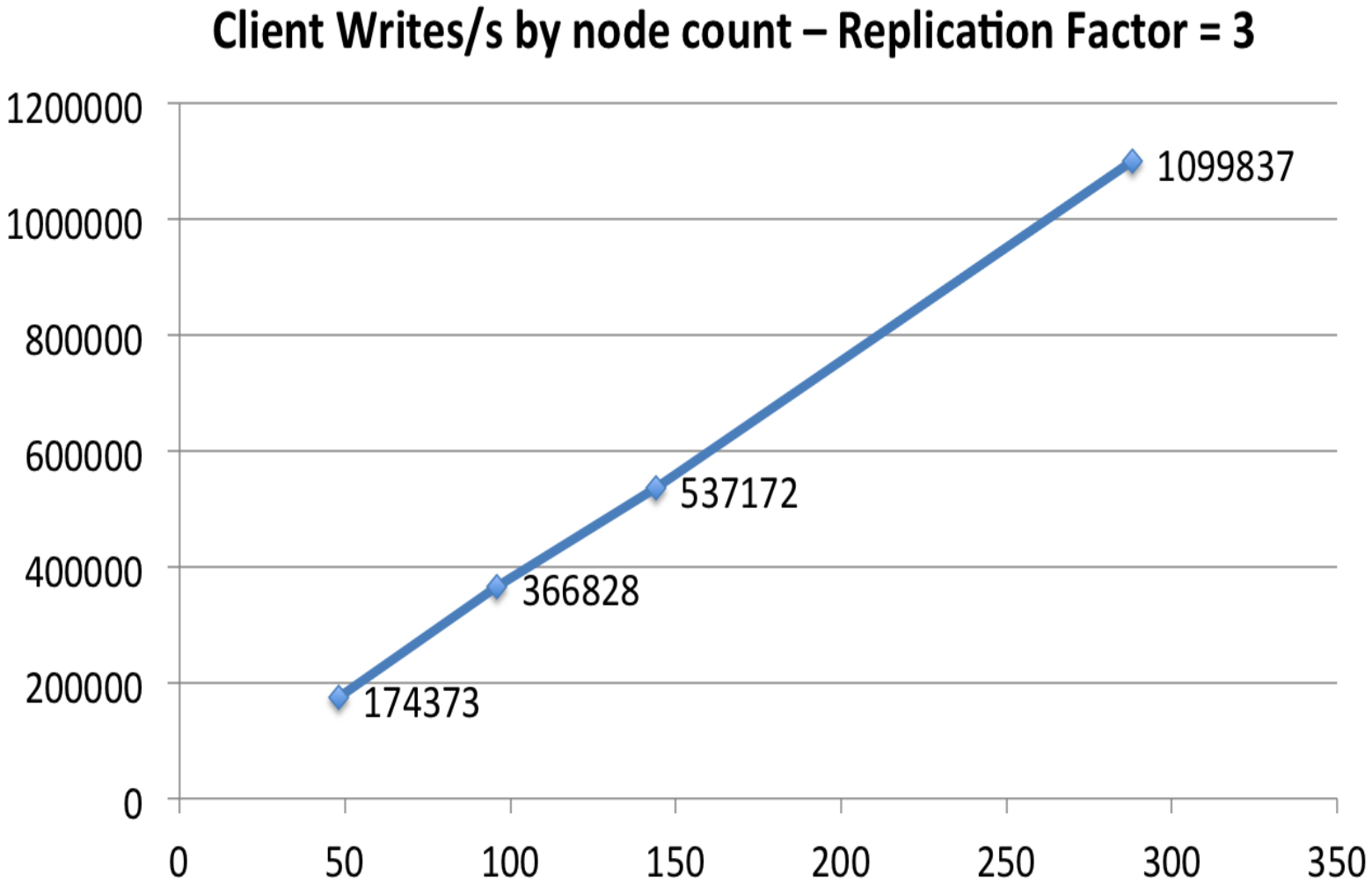
- How?
- Is this the right question to ask?
- What does “Big Data” have to say?

Block chain: what

"A block chain is a transaction database shared by all nodes participating in a system based on the Bitcoin protocol."

https://en.bitcoin.it/wiki/Block_chain

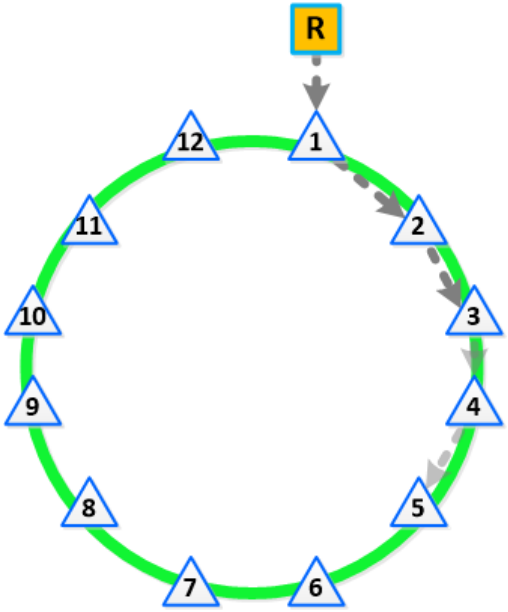
From Big Data: Cassandra DB. Scale-up linearity!

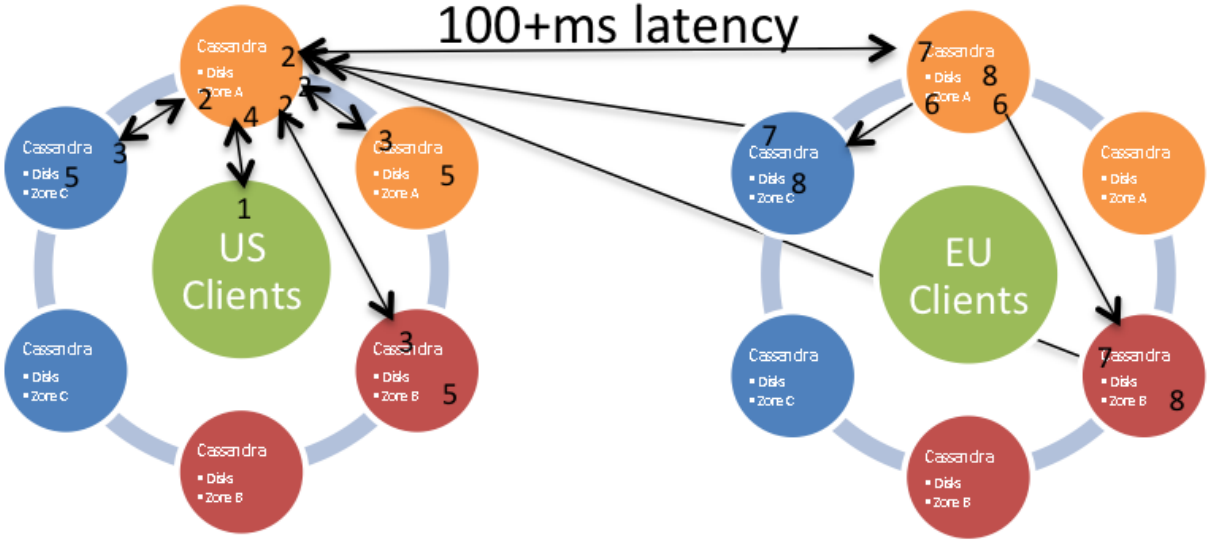


http://1.bp.blogspot.com/-ZFtW7MFMqZQ/TrG5ujuDGdI/AAAAAAAAAAW w/heceeMD50x4/s1600/scale.png

NETFLIX

Cassandra DB: How it works





http://stevenpoitras.com/the-nutanix-bible/

http://1.bp.blogspot.com/-Hsr6O8pwyzU/TrG5didDPjI/AAAAAAAAAWM/A3vL3wMkQgw/s1600/global.png

Cassandra DB: Consensus

"Paxos Algorithm"

(more later)

Block chain DB: Consensus

"Bitcoin uses the block chain algorithm to achieve distributed **consensus** on who owns what coins."

https://en.bitcoin.it/wiki/Alternative_chain

A walk down history lane: a gauntlet was thrown down

"Can you implement a distributed database that can tolerate the failure of any number of its processes (possibly all of them) without losing consistency, and that will resume normal behavior when more than half the processes are again working properly?"

-Leslie Lamport to colleagues, 1980

http://research.microsoft.com/en-us/um/people/lamport/pubs/pubs.html (referring to Paxos)

Gauntlet 2, Byzantine problem

Before [1], it was generally assumed that a three-processor system could tolerate one faulty processor. This paper shows that "Byzantine" faults, in which a faulty processor sends inconsistent information to the other processors, can defeat any traditional three-processor algorithm. (The term Byzantine didn't appear until [46].) In general, 3n+1 processors are needed to tolerate n faults. However, if digital signatures are used, 2n+1 processors are enough.

This paper introduced the problem of handling Byzantine faults.

I think it also contains the first precise statement of the consensus problem.

http://research.microsoft.com/en-us/um/people/lamport/pubs/pubs.html

-Lesley Lamport

[1] L. Lamport et al, Reaching Agreement in the Presence of Faults , J. ACM *27(2), April 1980*

Towards Practical Solutions to Byzantine Generals Problem

"A fault-tolerant file system called Echo was built at SRC in the late 80s. The builders **claimed that it would maintain consistency despite any number of non-Byzantine faults, and would make progress if any majority of the processors were working**. As with most such systems, it was quite simple when nothing went wrong, but had a complicated algorithm for handling failures based on taking care of all the cases that the implementers could think of. I decided that what they were trying to do was impossible, and set out to prove it."

http://research.microsoft.com/en-us/um/people/lamport/pubs/pubs.html

The Practical Solution to Byzantine Generals Problem

"I decided that what they were trying to do was impossible, and set out to prove it.

Instead, **I discovered the Paxos algorithm... At the heart of the algorithm is a three-phase consensus protocol.**

..to my knowledge, Paxos contains the first three-phase commit algorithm that is a real algorithm, with a clearly stated correctness condition and a proof of correctness..”

http://research.microsoft.com/en-us/um/people/lamport/pubs/pubs.html

L. Lamport et al, "The Byzantine Generals Problem“, ACM Transactions on Programming Languages and Systems 4 (3): 382–401, July 1982

Extensions to Paxos

“Byzantine Paxos[8][10] adds an extra message (Verify) which acts to distribute knowledge and verify the actions of the other processors”

http://en.wikipedia.org/wiki/Paxos_(computer_science)#Byzantine_Paxos

[8] Lamport, Leslie (2005). "Fast Paxos".
[10] Castro, Miguel (2001). "Practical Byzantine Fault Tolerance".

Opinions on Paxos

“there is only one consensus protocol, and that’s Paxos”
-Mike Burrows, inventor of Chubby service at Google

“all other approaches are just broken versions of Paxos.”

“The Paxos protocol .. is famously subtle and a bit difficult to .. it’s clear that a good consensus protocol is surprisingly hard to find.”

http://the-paper-trail.org/blog/consensus-protocols-two-phase-commit/

Production Use of Paxos

- Google – all products that use BigTable or Spanner (search, analytics, email, ..)
- Clustrix - distributed SQL DB
- **Apache Cassandra – distributed NoSQL**
- FoundationDB – distributed NewSQL
- Neo4j HA – graph DB
- (and most other modern distributed DBs!)
- Heroku / Salesforce, Microsoft, IBM, …

[wikipedia, more]

Explaining Paxos: Two Phase Commit (2PC)

- (1) “Do you?” (2) “I do!” “I do!”
- Problem: inconsistent result if node failure after “Do you?”

http://the-paper-trail.org/blog/consensus-protocols-two-phase-commit/

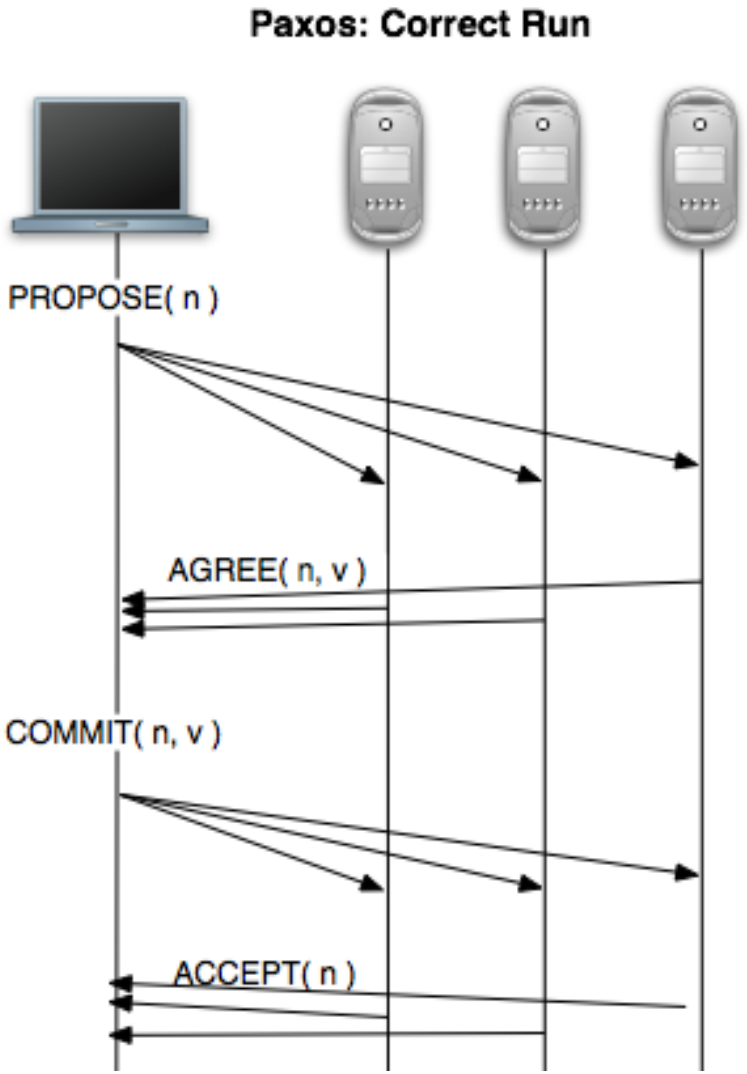
Explaining Paxos: Three Phase Commit (3PC)

- Break the “I do” part into two phases
  - Prepare to commit. (“Don’t listen to any more do-you’s for now”)
  - Commit. “I do!”

Can still fail: one site is in “prepared to commit” when another is not.

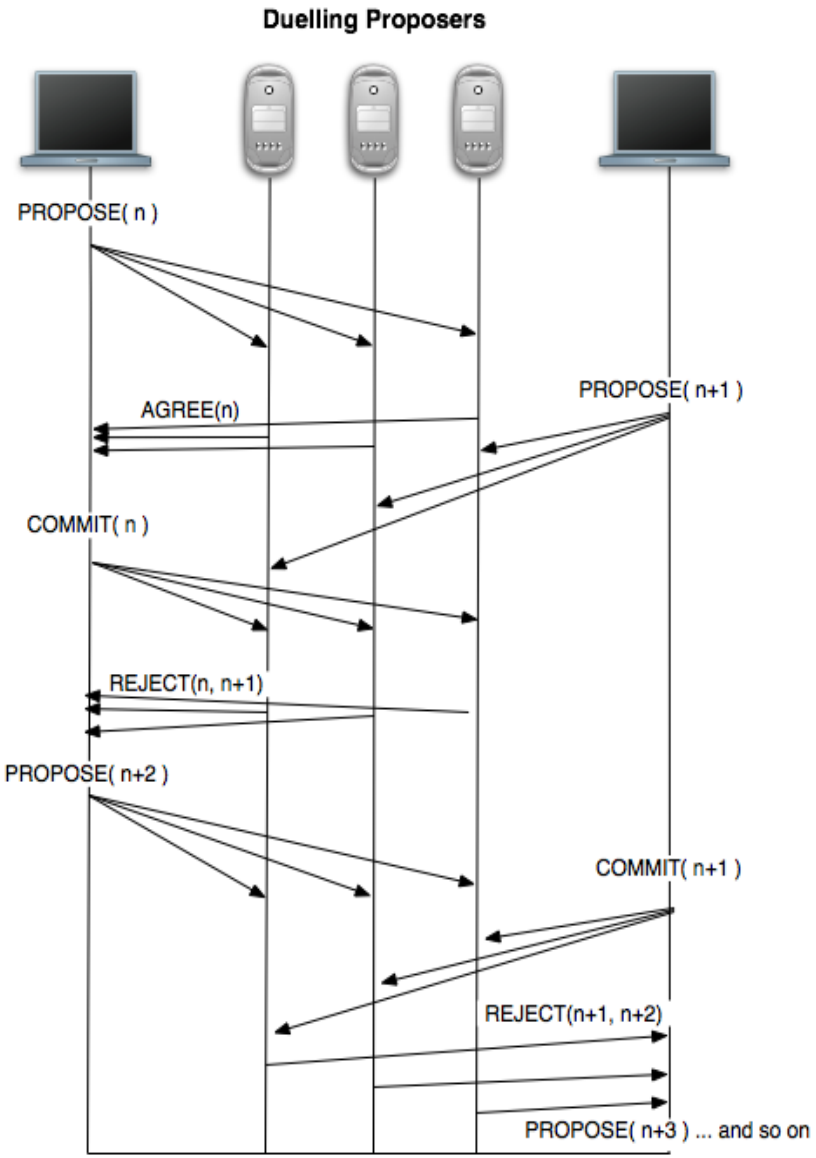
http://the-paper-trail.org/blog/consensus-protocols-three-phase-commit/

Explaining Paxos: The Algorithm



http://the-paper-trail.org/blog/consensus-protocols-paxos/

Explaining Paxos: The Algorithm



http://the-paper-trail.org/blog/consensus-protocols-paxos/

Block chain: A design decision…

"The block chain is broadcast to all nodes on the networking [sic] using a flood protocol"

[https://en.bitcoin.it/wiki/Block_chain]

(Very inefficient!)

My thoughts:
Two approaches to scale up

- **Big data-fy the blockchain**
  - Builds on man-decades of work
  - Significant scalability hurdles?

<or>

- **Blockchain-ify big data**
  - Builds on man-centuries (millennia?) of work
  - Scalability challenges already resolved
  - Needs distributed control (which isn't easy either!)

Questions I have

- Why doesn't the Bitcoin community talk more about Paxos? (Yet it does talk about Byzantine Generals)

- Why isn't the blockchain itself partially distributed? (in the sense that pieces of it are sharded throughout the network, rather than a full duplicate everywhere)
  - Am I missing something? Is there something fundamentally wrong with "blockchain-ifying big data?"

Summary

- The blockchain is a DB, as are modern “Big Data” NoSQL and NewSQL DBs. They’re all distributed.
- Distributing a DB by making a full copy on every node scales extremely poorly.
- Distributed DBs need a consensus algorithm. Posed as the Byzantine Generals problem (Lesley Lamport)
- Most modern “Big Data” DBs use the **Paxos** Algorithm. “all other approaches are just broken versions of Paxos.”
- Open Q’s summary:
  - Paxos <-> BTC relation?
  - **Can we blockchain-ify big data?**