Distributed Consensus: Making Impossible Possible

QCon London
Tuesday 29/3/2016

Heidi Howard
PhD Student @ University of Cambridge
heidi.howard@cl.cam.ac.uk
@heidiann360
defined to be the largest vote \( v \) in Votes\( (B) \) cast by \( p \) with \( v_{\text{bal}} < b \), or to be null\( p \) if there was no such vote. Since null\( p \), is smaller than any real vote cast by \( p \), this means that MaxVote\( (b, p, B) \) is the largest vote in the set

\[
\{ v \in \text{Votes}(B) : (v_{\text{bal}} = p) \land (v_{\text{bal}} < b) \} \cup \{ \text{null}\ p \}
\]

For any nonempty set \( Q \) of priests, \( \text{MaxVote}(b, Q, B) \) was defined to equal the maximum of all votes \( \text{MaxVote}(b, p, B) \) with \( p \) in \( Q \).

Conditions B1(B)–B3(B) are stated formally as follows.

\[ B \]

\[ B1(p) \quad \text{The Part-Time Parliament} \]

\[ B \]

\[ B2(p) \quad \text{Associated variables: prebal}(p), \text{decr}(p) \]

\[ B3(p) \quad \text{Associated variables: prevotes}(p) \]

Although implies th numbers \( v \) To show \( B1(B) \leq B \) is for th

Lemma 15(p) \( \Box \quad \text{Associated variables: quorum}(p), \text{voters}(p), \text{decr}(p) \]

\[ \forall \exists B : B \quad \text{Associated variables: owner}\]

Proof of For any b decre dif

16 \( \Box \quad \text{Associated variables: B} \]

\[ \forall \exists B : B \quad \text{Associated variables: M} \]

To prove the Paxon

\[ B_{\text{bal}} \leq B \quad \text{Associated variables: A} \]

1. Choose \( B_{\text{bal}} \leq B \)

2. \( B_{\text{bal}} \leq B \)

3. \( B_{\text{bal}} \leq B \)

The Paxon had to prove that \( J \) satisfies the three conditions given above. The first condition, that \( J \) holds initially, requires checking that each conjunct is true for the initial values of all the variables. While not stated explicitly, these initial values can be inferred from the variables' descriptions, and checking the first condition is straightforward. The second condition, that \( J \) implies consistency, follows from J, the first conjunct of 16, and Theorem 1. The hard part was proving the third condition, the invariance of \( I \), which meant proving that \( I \) is left true by every action. This condition is proved by showing that, for each conjunct of \( I \), executing any action when \( I \) is true leaves that conjunct true. The proofs are sketched below.

\[ I1(p) \]

A Hundred Impossibility Proofs for Distributed Computing

Nancy A. Lynch

The Part-Time Parliament

\[ B \]

\[ I3(p) \quad \text{Associated variables: prebal}(p), \text{decr}(p) \]

\[ \text{Associated variables: prevotes}(p) \]

Although implies th

\[ I4(p) \quad \text{Associated variables: prevotes}(p) \]

\[ \text{Associated variables: quorum}(p), \text{voters}(p), \text{decr}(p) \]

1 Introduction

This talk is about impossibility results in the area of distributed computing. In this category, I include not just results that say that a particular task cannot be accomplished, but also lower bound results, which say that a task cannot be accomplished within a certain bound on cost.

I started out with a simple plan for preparing this talk: I would spend a couple of weeks reading all the impossibility proofs in our field, and would categorise them according to the ideas used. Then I would make wise and general observations, and try to predict where the future of this area is headed. That turned out to be a bit too ambitious; there are many more such results than I thought. Although it is often hard to say what constitutes a "different result", I managed to count over 100 such impossibility proofs! And my search wasn’t even very systematic or exhaustive.

It’s not quite as hopeless to understand this area as it might seem from the number of papers. Although there are 100 different results, there aren’t 100 different ideas. I thought I could contribute something by identifying some of the commonality among the different results.

So what I will do in this talk will be an incomplete version of what I originally intended. I will give you...
What is Consensus?

“The process by which we reach agreement over system state between unreliable machines connected by asynchronous networks”
Why?

- Distributed locking
- Banking
- Safety critical systems
- Distributed scheduling and coordination

Anything which requires guaranteed agreement
A walk through history

We are going to take a journey through the developments in distributed consensus, spanning 3 decades.

We are going to search for answers to questions like:

• how do we reach consensus?
• what is the best method for reaching consensus?
• can we even reach consensus?
• what’s next in the field?
FLP Result

off to a slippery start

Impossibility of distributed consensus with one faulty process

Michael Fischer, Nancy Lynch and Michael Paterson

ACM SIGACT-SIGMOD Symposium on Principles of Database Systems

1983
FLP

We cannot guarantee agreement in an asynchronous system where even one host might fail.

Why?

We cannot reliably detect failures. We cannot know for sure the difference between a slow host/network and a failed host

NB: We can still guarantee safety, the issue limited to guaranteeing liveness.
Solution to FLP

In practice:

We accept that sometimes the system will not be available. We mitigate this using timers and backoffs.

In theory:

We make weaker assumptions about the synchrony of the system e.g. messages arrive within a year.
Paxos
Lamport’s original consensus algorithm

The Part-Time Parliament
Leslie Lamport
ACM Transactions on Computer Systems
May 1998
Paxos

The original consensus algorithm for reaching agreement on a single value.

- two phase process: prepare and commit
- majority agreement
- monotonically increasing numbers
Paxos Example - Failure Free
Incoming request from Bob
Phase 1

1

Promise (13) ?

2

3

P: 13
C:

B

P:
C:
Phase 1
Phase 2

Commit (13, B)?
Phase 2
Bob is granted the lock
Paxos Example - Node Failure
Incoming request from Bob

Phase 1

Promise (13)?
Phase 1

Diagram showing a network of nodes labeled 1, 2, and 3. Node 1 and node 2 are connected with arrows labeled "OK." Node 1 is connected to node 3 with an arrow labeled "OK." Node 3 is locked and has a key symbol.

Nodes:

1

2

3

Node Labels:
P: 13
C:

Phase 1
Commit (13, B)?
Phase 2
Alice would also like the lock.
Alice would also like the lock
Phase 1

P: 13
C: 13, B

P: 22
C:

Promise (22)?
Phase 1

OK(13, B)
Phase 2

Commit (22, B) ?

P: 22, C: 13, B

P: 22, C: 22, B

P: 13, C: 13, B

A

B

1

2
Phase 2

- **Phase 1**
  - Node 1: P: 22, C: 22
  - Node 2: P: 22, C: 22
  - Connection: OK

- **Phase 2**
  - Node 1: P: 22, C: 22
  - Node 2: P: 22, C: 22
  - Connection: NO

- **Errors**
  - Node 1: Red X (P: 13, C: 13)
  - Node 2: Red X (P: 13, C: 13)
Paxos Example - Conflict
Phase 1 - Bob
Phase 1 - Alice
Paxos Summary

Clients much wait two round trips (2 RTT) to the majority of nodes. Sometimes longer.

The system will continue as long as a majority of nodes are up
Multi-Paxos
Lamport’s leader-driven consensus algorithm

Paxos Made Moderately Complex
Robbert van Renesse and Deniz Altinbuken
ACM Computing Surveys
April 2015

Not the original, but highly recommended
Multi-Paxos

Lamport’s insight:

Phase 1 is not specific to the request so can be done before the request arrives and can be reused.

Implication:

Bob now only has to wait one RTT
State Machine Replication
fault-tolerant services using consensus

Implementing Fault-Tolerant Services Using the State Machine Approach: A Tutorial
Fred Schneider
ACM Computing Surveys
1990
State Machine Replication

A general technique for making a service, such as a database, fault-tolerant.
CAP Theorem
You cannot have your cake and eat it

CAP Theorem
Eric Brewer
Presented at Symposium on Principles of Distributed Computing, 2000
Consistency, Availability & Partition Tolerance - Pick Two
Paxos Made Live
How google uses Paxos

Paxos Made Live - An Engineering Perspective
Tushar Chandra, Robert Griesemer and Joshua Redstone
ACM Symposium on Principles of Distributed Computing
2007
Paxos Made Live

Paxos made live documents the challenges in constructing Chubby, a distributed coordination service, built using Multi-Paxos and SMR.
Isn’t this a solved problem?

“There are significant gaps between the description of the Paxos algorithm and the needs of a real-world system.

In order to build a real-world system, an expert needs to use numerous ideas scattered in the literature and make several relatively small protocol extensions.

The cumulative effort will be substantial and the final system will be based on an unproven protocol.”
Challenges

• Handling disk failure and corruption
• Dealing with limited storage capacity
• Effectively handling read-only requests
• Dynamic membership & reconfiguration
• Supporting transactions
• Verifying safety of the implementation
Fast Paxos
Like Multi-Paxos, but faster
Fast Paxos

**Paxos:** Any node can commit a value in 2 RTTs

**Multi-Paxos:** The leader node can commit a value in 1 RTT

But, what about any node committing a value in 1 RTT?
Fast Paxos

We can bypass the leader node for many operations, so any node can commit a value in 1 RTT.

However, we must either:

• reduce the number of failures we guarantee to tolerance, or

• increase the size of the quorum, or

• a combination of both
Egalitarian Paxos

Don’t restrict yourself unnecessarily

There Is More Consensus in Egalitarian Parliaments
Iulian Moraru, David G. Andersen, Michael Kaminsky
SOSP 2013

also see Generalized Consensus and Paxos
Egalitarian Paxos

The basis of SMR is that every replica of an application receives the same commands in the same order.

However, sometimes the ordering can be relaxed...
Partial Ordering

Total Ordering

C=1
→ B?
C=C+1
→ C?
B=0
→ B=C

→ C=1
→ C=C+1
→ C?
→ B=0
→ B=C
Many possible orderings
Egalitarian Paxos

Allow requests to be out-of-order if they are commutative.

Conflict becomes much less common.

Works well in combination with Fast Paxos.
Viewstamped Replication Revisited
the forgotten algorithm

Viewstamped Replication Revisited
Barbara Liskov and James Cowling
MIT Tech Report
MIT-CSAIL-TR-2012-021
Viewstamped Replication Revisited (VRR)

Interesting and well explained variant of SMR + Multi-Paxos.

Key features:

• Round robin leader election

• Dynamic Membership
Raft

Raft has taken the wider community by storm. Due to its understandable description

It’s another variant of SMR with Multi-Paxos.

Key features:

• Really strong leadership - all other nodes are passive

• Dynamic membership and log compaction
Ios

Why do things yourself, when you can delegate it?

to appear
Ios

The issue with leader-driven algorithms like Multi-Paxos, Raft and VRR is that throughput is limited to one node.

Ios allows a leader to safely and dynamically delegate their responsibilities to other nodes in the system.
Hydra

consensus for geo-replication

to appear
Hydra

Distributed consensus for systems which span multiple datacenters.

We use Ios for replication within the datacenter and a Egalitarian Paxos like protocol for across datacenters.

The system has a clear leader but most requests simply bypass the leader.
The road we travelled

- **2 impossibility results**: CAP & FLP
- **1 replication method**: State machine Replication
- **6 consensus algorithms**: Paxos, Multi-Paxos, Fast Paxos, Egalitarian Paxos, Viewstamped Replication Revisited & Raft
- **2 future algorithms**: Ios & Hydra
How strong is the leadership?

- Strong Leadership
- Leader driven
- Leader with Delegation
- Leader only when needed
- Leaderless

- Raft
- VRR
- Multi-Paxos
- Ios
- Hydra
- Fast Paxos
- Egalitarian Paxos
Who is the winner?

Depends on the award:

• Best for minimum latency: VRR
• Easier to understand: Raft
• Best for WANs (conflicts rare): Egalitarian Paxos
• Best for WANs (conflicts common): Fast Paxos
Future

1. More algorithms offering a compromise between strong leadership and leaderless

2. More understandable consensus algorithms

3. Achieving consensus is getting cheaper, even in challenging settings

4. Deployment with micro-services and unikernels

5. Self-scaling replication - adapting resources to maintain resilience level.
Stops we drove passed

We have seen one path through history, but many more exist.

• Alternative replication techniques e.g. chain replication and primary backup replication

• Alternative failure models e.g. nodes acting maliciously

• Alternative domains e.g. sensor networks, mobile networks, between cores
Summary

Do not be discouraged by impossibility results and dense abstract academic papers.

Consensus is useful and achievable.

Find the right algorithm for your specific domain.