Distributed Consensus: Making Impossible Possible

Heidi Howard
PhD Student @ University of Cambridge
heidi.howard@cl.cam.ac.uk
@heidiann360
hh360.user.srcl.net
Sometimes inconsistency is not an option

- Distributed locking
- Financial services/ blockchain
- Safety critical systems
- Distributed scheduling and coordination
- Strongly consistent databases

Anything which requires guaranteed agreement
What is Consensus?

“The process by which we reach agreement over system state between unreliable machines connected by asynchronous networks”
A Hundred Impossibility Proofs for Distributed Computing

Nancy A. Lynch *
Lab for Computer Science
MIT, Cambridge, MA 02139
lynnch@ids.cs.lcs.mit.edu

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 categorize 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 a tour of the impossibility results that I was able to collect. I apologize for not being comprehensive, and in particular for placing perhaps undue emphasis on results I have been involved in (but those are the ones I know best!). I will describe the techniques used, as well as give some historical perspective. I’ll intersperse this with my opinions and observations, and I’ll try to collect what I consider to be the most important of these at the end. Then I’ll make some suggestions for future work.

2 The Results

1 classified the impossibility results I found into the following categories: shared memory resource allocation, distributed consensus, shared registers, computing in rings and other networks, communication protocols, and miscellaneous.

2.1 Shared Memory Resource Allocation

This was the area that introduced me not only to the possibility of doing impossibility proofs for distributed computing, but to the entire distributed computing research area.

In 1976, when I was at the University of Southern California, Armin Creners and Tom Hibbard were playing with the problem of mutual exclusion (or allocation of one resource) in a shared memory environment. In the environment they were considering, a group of asynchronous processes communicate via shared memory, using operations such as read and write or test-and-set.

The previous work in this area had consisted of a series of papers by Dijkstra [38] and others, each presenting a new algorithm guaranteeing mutual exclusion, along with some other properties such as progress and fairness. The properties were specified somewhat loosely; there was no formal model used for...
A walk through history

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

Note: 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.
Viewstamped Replication
the forgotten algorithm

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

Not the original from 1988, but recommended
In my view, the pioneer on the field of consensus.

Let one node be the ‘master’, rotating when failures occur. Replicate requests for a state machine.

Now considered a variant of SMR + Multi-Paxos.
Paxos
Lamport’s consensus algorithm

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

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

- two phase process: promise and commit
- each requiring majority agreement (aka quorums)
- 2 RRTs to agreement a single value
Paxos Example - Failure Free
Incoming request from Bob
Phase 1

P: 13
C:

Promise (13) ?

1

Promise (13) ?

2

3

B

P: 13
C:
Phase 2

Commit (13, B) ?

Commit (13, B) ?

P: 13
C: 13,
B

P: 13
C: 13,
B
Phase 2

P: 13
C: 13, B

OK

1

OK

2

OK

3

P: 13
C: 13, B

Phase 2
Bob is granted the lock
Paxos Example - Node Failure
Incoming request from Bob

Phase 1

Promise (13) ?

1

Promise (13) ?

2

P: 13
C:

3
Commit (13, B) ?

Phase 2
Phase 2
Alice would also like the lock.
Alice would also like the lock.
Phase 1

P: 13  C: 13, B

Promise (22) ?

P: 22  C:

1 ➔ 2

P: 13  C: 13, B
Phase 1

P: 22
C: 13,

OK(13, B)

P: 22
C:

Phase 1
Commit (22, B) ?
Phase 2
Paxos Example - Conflict
Phase 1 - Bob
Phase 1 - Alice

1

2

3

P: 21
C:

P: 21
C:

Phase 1 - Alice
Phase 1 - Bob

A

B

P: 33
C:

1

2

3

P: 33
C:
Paxos

Clients must 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
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.”
Paxos Made Live

Paxos made live documents the challenges in constructing Chubby, a distributed coordination service, built using Multi-Paxos and SMR.
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
Leslie Lamport
Microsoft Research Tech Report
MSR-TR-2005-112
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 increase the size of the quorum.
Zookeeper
The open source solution

Zookeeper: wait-free coordination for internet-scale systems
Hunt et al
USENIX ATC 2010

Code: zookeeper.apache.org
Zookeeper

Consensus for the masses.

It utilizes and extends Multi-Paxos for strong consistency.

Unlike “Paxos made live”, this is clearly discussed and openly available.
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…
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.
Raft Consensus
Paxos made understandable

In Search of an Understandable Consensus Algorithm
Diego Ongaro and John Ousterhout
USENIX Annual Technical Conference
2014
Raft

Raft has taken the wider community by storm. Largely, due to its understandable description.

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

Key features:

• Really strong leadership - all other nodes are passive

• Various optimizations - e.g. dynamic membership and log compaction
Startup/Restart

Follower

Timeout

Candidate

Step down

Step down

Win

Leader

Timeout
Why do things yourself, when you can delegate it?
The issue with leader-driven algorithms like Viewstamp Replication, Multi-Paxos, Zookeeper and Raft 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.
Flexible Paxos
Paxos made scalable

Flexible Paxos: Quorum intersection revisited
Heidi Howard, Dahlia Malkhi, Alexander Spiegelman
ArXiv:1608.06696
Majorities are not needed

Usually, we use require majorities to agree so we can guarantee that all quorums (groups) intersect.

This work shows that not all quorums need to intersect. Only the ones used for replication and leader election.

This applies to all algorithms in this class: Paxos, Viewstamped Replication, Zookeeper, Raft etc..
The road we travelled

• 2 theoretical results: FLP & Flexible Paxos

• 2 popular ideas: CAP & Paxos made live

• 1 replication method: State machine Replication

• 8 consensus algorithms: Viewstamped Replication, Paxos, Multi-Paxos, Fast Paxos, Zookeeper, Egalitarian Paxos, Raft & Ios
How strong is the leadership?

- Strong Leadership
  - Raft
  - Viewstamped Replication
  - Multi-Paxos
  - Zookeeper

- Leader driven
  - Leader only when needed

- Leader with Delegation
  - Fast Paxos
  - Egalitarian Paxos

- Leaderless
  - Paxos
Who is the winner?

Depends on the award:

• Best for minimum latency: Viewstamped Replication

• Most widely used open source project: Zookeeper

• Easiest to understand: Raft

• Best for WANs: Egalitarian Paxos
Future

1. More scalable and performant consensus algorithms utilizing Flexible Paxos.

2. A clearer understanding of consensus and better explained consensus algorithms.

3. Achieving consensus in challenge settings such as geo-replicated systems.
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.

Don’t give up on consistency. Consensus is achievable, even performant and scalable (if done correctly)

Find the right algorithm for your specific domain.

heidi.howard@cl.cam.ac.uk
@heidiann360