Distributed Consensus: Making Impossible Possible

Heidi Howard
PhD Student @ University of Cambridge
heidi.howard@cl.cam.ac.uk
@heidiann360
A Hundred Impossibility Proofs for Distributed Computing

Nancy A. Lynch

A Hundred Impossibility Proofs for Distributed Computing

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 speak 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 giving 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

I 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 category contains the subset of problems that can be done in shared memory. For each problem, we restrict the types of processors that are allowed, and the number of processors. The main results are the impossibility results.

In 1976, when I was at the University of Southern California, Armin Cremers 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 to 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

*Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the ACM copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Association for Computing Machinery. To copy otherwise, or to republish, requires a fee and/or specific permission.

© 1989 ACM 0-89791-326-4/89/0008/0011 $1.50

1
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: promise and commit
- majority agreement
- monotonically increasing numbers
Paxos Example - Failure Free
Incoming request from Bob
Phase 1
Phase 1

Diagram:

- Node 1
  - State: OK
- Node 2
  - State: OK
- Node 3
  - State: OK

Legend:
- **P**: Process
- **C**: Control
- **13**: Value

Notes:
- The diagram depicts the flow of states between nodes 1, 2, and 3, indicating a successful process flow in Phase 1.
Phase 2

Commit (13, B) ?

Commit (13, B) ?

Commit (13, B) ?
Phase 2

1

OK

2

OK

3

P: 13
C: 13,

B

P: 13
C: 13,

B

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 → 3

Promise (13) ?

3 → 2

P: 13
C:
Phase 1

1 -- OK --> 3

2 -- OK --> 3

P: 13
C:

P: 13
C:
Phase 2

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

1. Node 1 sends a message to node 2 with the values P: 22 and C: 13.
2. Node 2 sends a message to node 1 with the values P: 22, C:.
3. Node 1 sends a message to node 2 with the values P: 13, C: 13.

The communication is shown in the diagram with arrows indicating the direction of the messages.
Phase 2

Commit (22, B) ?

P: 22, C: 13, B

P: 22, C: 22, B

P: 13, C: 13, B

A

2

1

Red X
Paxos Summary

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
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
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 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…
C = 1 → B? → C = C + 1 → C? → B = 0 → B = C

Total Ordering

Partial Ordering
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 (VRR)

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

Key features:

- Round robin leader election
- Dynamic Membership
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. 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
Startup/Restart

Follower ➔ Timeout ➔ Candidate ➔ Win ➔ Leader

Step down

Timeout

Step down
Ios

Why do things yourself, when you can delegate it?

to appear
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?
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.
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.

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