Reaching reliable agreement in an unreliable world

Heidi Howard
heidi.howard@cl.cam.ac.uk
@heidiann

Research Students Lecture Series
13th October 2015
Introducing Alice

Alice is new graduate from Cambridge off to the world of work.

She joins a cool new start up, where she is responsible for a key value store.
Single Server System
Single Server System

Client 1 → Server

Client 2

Client 3

A? 7
Single Server System

Client 1

Client 2

Client 3

Server

A 7
B 2;3

B=3, OK
Single Server System

Client 1

Client 2

Client 3

Server

\[\begin{array}{cc}
A & 7 \\
B & 2-3 \\
\end{array}\]

3

B?
Single Server System

Pros

• linearizable semantics

• durability with write-ahead logging

• easy to deploy

• low latency (1 RTT in common case)

• partition tolerance with retransmission & command cache

Cons

• system unavailable if server fails

• throughput limited to one server
Backups

A? 7
B 2

Backup 1

A 7
B 2

Primary

A? 7
B 2

Client 1

A? 7
B 2

Client 2

\textit{aka} Primary backup replication
Backups

aka Primary backup replication
Backups

aka Primary backup replication

Client 1

Client 2

Backup 1

Backup 1

Backup 1

Primary

A 7
B 1

A 7
B 1

A 7
B 1

A 7
B 1

OK

OK

OK

OK
Big Gotcha

We are assuming total ordered broadcast
Totally Ordered Broadcast

(aka atomic broadcast) the guarantee that messages are received reliably and in the same order by all nodes.
Requirements

• **Scalability** - High throughout processing of operations.

• **Latency** - Low latency commit of operation as perceived by the client.

• **Fault-tolerance** - Availability in the face of machine and network failures.

• **Linearizable semantics** - Operate as if a single server system.
Doing the Impossible
CAP Theorem

Pick 2 of 3:

• Consistency
• Availability
• Partition tolerance

Proposed by Brewer in 1998, still debated and regarded as misleading. [Brewer’12] [Kleppmann’15]
It is impossible to guarantee consensus when messages may be delay if even one node may fail. [JACM’85]
Consensus is impossible

A Hundred Impossibility Proofs for Distributed Computing

Nancy A. Lynch
Lab for Computer Science
MIT, Cambridge, MA 02139
lynch@ds.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 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 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 1978, when I was at the University of Southern California, Armijn Cremer 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 [36] 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
Aside from Simon PJ

Don’t drag your reader or listener through your blood strained path.

Simon Peyton Jones
Paxos

Paxos is at the foundation of (almost) all distributed consensus protocols.

It is a general approach of using two phases and majority quorums.

It takes much more to construct a complete fault-tolerance distributed systems.

Leslie Lamport
Beyond Paxos

• Replication techniques - mapping a consensus algorithm to a fault tolerance system

• Classes of operations - handling of read vs writes, operating on stale data, soft vs hard state

• Leadership - fixed, variable or leaderless, multi-paxos, how to elect a leader, how to discover a leader

• Failure detectors - heartbeats & timers

• Dynamic membership, Log compaction & GC, sharding, batching etc
Consensus is hard
A raft in the sea of confusion
Case Study: Raft

Raft is the understandable replication algorithm.

Provides linearisable client semantics, 1 RTT best case latency for clients.

A complete(ish) architecture for making our application fault-tolerance.

Uses SMR and Paxos
State Machine Replication

Client

Server

Server

Server

A 7
B 2

A 7
B 2

A 7
B 2

B=3
State Machine Replication

A 7
B 2
Server

A 7
B 2
Server

A 7
B 2
Server

B=3

Client
State Machine Replication
Leadership

Startup/Restart

Follower

Timeout

Candidate

Step down

Step down

Win

Leader

Timeout
Ordering

Each node stores its own perspective on a value known as the term.

Each message includes the sender’s term and this is checked by the recipient.

The term orders periods of leadership to aid in avoiding conflict.

Each has one vote per term, thus there is at most one leader per term.
Ordering

ID: 1

ID: 2

ID: 3
Leadership

Startup/Restart

Follower

Timeout

Candidate

Step down

Step down

Step down

Win

Leader
Vote for me in term 1!
ID: 1
Term: 1
Vote: 4

ID: 5
Term: 1
Vote: 4

ID: 2
Term: 1
Vote: 4

ID: 4
Term: 1
Vote: me

ID: 3
Term: 1
Vote: 4

Ok!
Leader fails
Vote for me in term 2!

- ID: 5
  - Term: 2
  - Vote: me

- ID: 1
  - Term: 2
  - Vote: 5

- ID: 2
  - Term: 1
  - Vote: 4

- ID: 4
  - Term: 1
  - Vote: me

- ID: 3
  - Term: 1
  - Vote: 4
Vote for me in term 2!
Leadership

Startup/Restart

Follower

Step down

Candidate

Step down

Timeout

Leader

Step down

Win

Timeout
Vote for me in term 3!
Replication

Each node has a log of client commands and an index into this representing which commands have been committed.
Simple Replication

<table>
<thead>
<tr>
<th>ID: 1</th>
<th>A</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>B</td>
<td>2</td>
</tr>
</tbody>
</table>

A=4

<table>
<thead>
<tr>
<th>ID: 2</th>
<th>A</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>B</td>
<td>2</td>
</tr>
</tbody>
</table>

A=4

<table>
<thead>
<tr>
<th>ID: 3</th>
<th>A</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>B</td>
<td>2</td>
</tr>
</tbody>
</table>

A=4
Simple Replication

ID: 1

\[
\begin{array}{c|c}
A & 4 \\
B & 2 \\
\end{array}
\]

A=4

ID: 2

\[
\begin{array}{c|c|c}
A & 4 & B=7 \\
B & 2 & \\
\end{array}
\]

A=4

ID: 3

\[
\begin{array}{c|c}
A & 4 \\
B & 2 \\
\end{array}
\]

A=4
Simple Replication

ID: 1

ID: 2

ID: 3
Simple Replication

ID: 1

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>4</td>
</tr>
<tr>
<td>B</td>
<td>2</td>
</tr>
</tbody>
</table>

A=4 | B=7

ID: 2

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>4</td>
</tr>
<tr>
<td>B</td>
<td>7</td>
</tr>
</tbody>
</table>

A=4 | B=7

ID: 3

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>4</td>
</tr>
<tr>
<td>B</td>
<td>2</td>
</tr>
</tbody>
</table>

A=4 | B=7
Simple Replication

ID: 1

<table>
<thead>
<tr>
<th>A</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>7</td>
</tr>
</tbody>
</table>

A=4  B=7

ID: 2

<table>
<thead>
<tr>
<th>A</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>7</td>
</tr>
</tbody>
</table>

A=4  B=7

ID: 3

<table>
<thead>
<tr>
<th>A</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>7</td>
</tr>
</tbody>
</table>

A=4  B=7
## Simple Replication

<table>
<thead>
<tr>
<th>ID: 1</th>
<th>ID: 2</th>
<th>ID: 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>![Table ID: 1](A=4, B=7)</td>
<td>![Table ID: 2](A=4, B=7)</td>
<td>![Table ID: 3](A=4, B=7)</td>
</tr>
</tbody>
</table>

- **Simple Replication**

- A = 4
- B = 7

- B?
Simple Replication

<table>
<thead>
<tr>
<th>ID: 1</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>4</td>
<td>A=4</td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>7</td>
<td>B=7</td>
<td>B?</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ID: 2</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>4</td>
<td>A=4</td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>7</td>
<td>B=7</td>
<td>B?</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ID: 3</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>4</td>
<td>A=4</td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>7</td>
<td>B=7</td>
<td>B?</td>
</tr>
</tbody>
</table>
Simple Replication

<table>
<thead>
<tr>
<th>ID: 1</th>
<th>A</th>
<th>4</th>
<th>A=4</th>
<th>B=7</th>
<th>B?</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID: 2</td>
<td>A</td>
<td>4</td>
<td>A=4</td>
<td>B=7</td>
<td>B?</td>
</tr>
<tr>
<td>ID: 3</td>
<td>A</td>
<td>4</td>
<td>A=4</td>
<td>B=7</td>
<td>B?</td>
</tr>
</tbody>
</table>
## Catchup

<table>
<thead>
<tr>
<th>ID: 1</th>
<th></th>
<th>A=4</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>A</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>B</td>
<td>2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ID: 2</th>
<th></th>
<th>A=4</th>
<th>B=7</th>
<th>B?</th>
<th>A=6</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>A</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>B</td>
<td>7</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ID: 3</th>
<th></th>
<th>A=4</th>
<th>B=7</th>
<th>B?</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>A</td>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>B</td>
<td>7</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Catchup

<table>
<thead>
<tr>
<th>ID: 1</th>
<th>A</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>B</td>
<td>2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ID: 2</th>
<th>A</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>B</td>
<td>7</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ID: 3</th>
<th>A</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>B</td>
<td>7</td>
</tr>
</tbody>
</table>

A=4 | B=7 | B? | A=6

:A(
## Catchup

| ID: 1 |   | A=4 | :(
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>2</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ID: 2</th>
<th>A=4</th>
<th>B=7</th>
<th>B?</th>
<th>A=6</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>7</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ID: 3</th>
<th>A=4</th>
<th>B=7</th>
<th>B?</th>
<th>A=6</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>7</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Catchup

**ID: 1**

<table>
<thead>
<tr>
<th>A</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>2</td>
</tr>
</tbody>
</table>

A=4

:)

**ID: 2**

<table>
<thead>
<tr>
<th>A</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>7</td>
</tr>
</tbody>
</table>

A=4 | B=7 | B? | A=6

**ID: 3**

<table>
<thead>
<tr>
<th>A</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>7</td>
</tr>
</tbody>
</table>

A=4 | B=7 | B? | A=6
Catchup

<table>
<thead>
<tr>
<th>ID: 1</th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A=4</td>
<td>B=7</td>
<td>B?</td>
<td>A=6</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ID: 2</th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>7</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A=4</td>
<td>B=7</td>
<td>B?</td>
<td>A=6</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ID: 3</th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>7</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A=4</td>
<td>B=7</td>
<td>B?</td>
<td>A=6</td>
<td></td>
</tr>
</tbody>
</table>
Evaluation

- The leader is a serious bottleneck -> limited scalability
- Can only handle the failure of a minority of nodes
- Some rare network partitions render protocol in livelock
Beyond Raft
Case Study: Tango

Tango is designed to be a scalable replication protocol.

It’s a variant of chain replication + Paxos.

It is leaderless and pushes more work onto clients.
Simple Replication

Server 1

Server 2

Server 3

Client 1

Client 2

Sequencer

Next: 1
Simple Replication

Server 1

Server 2

Server 3

Client 1

Client 2

Sequencer

A=4

A=4

A=4

B=2

B=2

A=4

B=5

A=4

A=4

B=5

Next: 2

Next?

1
Simple Replication

Server 1

Server 2

Server 3

Client 1

Client 2

Sequencer

Next: 2

B=5 @ 1

OK

B=5

A=4

A=4

A=4

A=4

B=2

B=2

B=2

B=2

B=5
Simple Replication

Server 1

Server 2

Sequencer

Server 3

A=4
B=5

A=4
B=5

A=4

A=4

B=5

B=5

B=5

A=4

B=2

B=2

Next: 2
Simple Replication

Server 1

Client 1

Client 2

Server 2

Server 3

Sequencer

A=4, B=5

A=4, B=5

A=4, B=5

A=4, B=5

A=4, B=5

B=5 @ 1

OK

Next: 2
Simple Replication

Server 1

0
A=4
B=5

1
0
A=4
B=5

1
A=4
B=5

Server 2

Server 3

Client 1

A 7
B 2

Client 2

A 4
B 5

Sequencer

Next: 2
Fast Read

Server 1

Client 1

Client 2

Sequencer

Server 2

Server 3

A=4  B=5

A=4  B=5

A=4  B=5

0  1

Check?

B?

Next: 2

64
Handling Failures

Sequencer is soft-state which can be reconstructed by querying head server.

Clients failing between receiving token and first read leaves gaps in the log. Clients can mark these as empty and space (though not address) is reused.

Clients may fail before completing a write. The next client can fill-in and complete the operation

Server failures are detected by clients, who initiate a membership change, using term numbers.
Evaluation

Tango is scalable, the leader is not longer the bottleneck.

Dynamic membership and sharding come for free with design.

High latency of chain replication
Next Steps
wait… we’re not finished yet!
Requirements

• **Scalability** - High throughout processing of operations.

• **Latency** - Low latency commit of operation as perceived by the client.

• **Fault-tolerance** - Availability in the face of machine and network failures.

• **Linearizable semantics** - Operate as if a single server system.
Many more examples

- **Raft [ATC’14]** - Good starting point, understandable algorithm from SMR + multi-paxos variant

- **VRR [MIT-TR’12]** - Raft with round-robin leadership & more distributed load

- **Tango [SOSP’13]** - Scalable algorithm for f+1 nodes, uses CR + multi-paxos variant

- **Zookeeper [ATC’10]** - Primary backup replication + atomic broadcast protocol (Zab [DSN’11])

- **EPaxos [SOSP’13]** - leaderless Paxos variant for WANs
Can we do even better?

• Self-scaling replication - adapting resources to maintain resilience level.

• Geo replication - strong consistency between wide area links

• Auto configuration - adapting timeouts and configure as network changes

• Integrated with unikernels, virtualisation, containers and other such deployment tech
Evaluation is hard

• few common evaluation metrics.
• often only one experiment setup is used.
• different workloads
• evaluation to demonstrate protocol strength
Lessons Learned

- Reaching consensus in distributed systems is doable
- Exploit domain knowledge
- Raft is a good starting point but we can do much better!

Questions?