Is there anybody out there?

Reactive Summit 2018 Montréal

Manuel Bernhardt / @elmanu
Disclaimer

[A career in distributed systems] is both exhilarating and frustrating. When things work, it's like a symphony. When they don't, it's like an eleventh-birthday party where half of the kids are on speed.

— Jeff Darcy, HekaFS (formerly CloudFS) project lead
• Helping companies to get started with reactive systems and to keep them running
• Lightbend consulting and training partner, focus on Akka (Cluster, Streams)
• Scuba-diver
Motivational quote

Life is a single player game. You’re born alone. You’re going to die alone. All of your interpretations are alone. All your memories are alone. You’re gone in three generations and no one cares. Before you showed up nobody cared. It’s all single player.

— Naval Ravikant
Key issues for building clusters

- **discovery**: who's there?
- **fault detection**: who's in trouble?
- **load balancing**: who can take up work?
Key issues for building clusters

- **discovery**: who's there?
- **fault detection**: who's in trouble?
- **load balancing**: who can take up work?

⇒ Group membership
Implementing group membership

1. Failure Detection
2. Dissemination
3. Consensus
Failure detection

Wish you were here (1975)
Failure detector key properties

- **Completeness**: crash-failure of any group member is detected by all non-faulty members
Failure detector key properties

- **Completeness**: crash-failure of any group member is detected by all non-faulty members.
- **Accuracy**: no non-faulty group member is declared as failed by any other non-faulty group member (no false positives).
Failure detector key properties

- **Completeness**: crash-failure of any group member is detected by all non-faulty members.

- **Accuracy**: no non-faulty group member is declared as failed by any other non-faulty group member (no false positives).

Also relevant in practice:

- speed
- network message load
Impossibility result

It is **impossible** for a failure detector algorithm to deterministically achieve **both completeness and accuracy** over an **asynchronous unreliable** network¹

---

¹ Chandra, Toueg: Unreliable failure detectors for reliable distributed systems (1996)
Trade-offs

• **Strong - Weak** completeness: *all / some* non-faulty members detect a crash

• **Strong - Weak** accuracy: there are *no / some* false-positives
Trade-offs

• **Strong - Weak** completeness: all / some non-faulty members detect a crash

• **Strong - Weak** accuracy: there are no / some false-positives

⇒ In practice most applications prefer strong completeness with a weaker form of accuracy
Failure Detector strategies

- **heartbeat**: no heartbeat = failure
- **ping**: no response = failure
Phi Adaptive Accrual Failure Detector

- has a cool name
- adaptive: adjusts to network conditions
- introduces the notion of *accrual* failure detection: suspicion value $\phi$ rather than boolean (trusted or suspected)
- made popular by Cassandra

---

Phi Adaptive Accrual Failure Detector

Example: master and worker processes

- $\phi(w) > 8 \Rightarrow$ stop sending new work to the node
- $\phi(w) > 10 \Rightarrow$ start to rebalance current tasks of the worker to other nodes
- $\phi(w) > 12 \Rightarrow$ remove the worker from the list of nodes

---


@elmanu - https://manuel.bernhardt.io
New Adaptive accrual Failure Detector

- much simpler to calculate suspicion level than Phi
- performs slightly better and *more adaptive*.

---

5 https://manuel.bernhardt.io/2017/07/26/a-new-adaptive-accrual-failure-detector-for-akka/
SWIM Failure Detector

As you swim lazily through the milieu,
The secrets of the world will infect you\(^6\)

- has both a dissemination and a failure detection component
- **scalable** membership protocol
- members are first suspected and not immediately flagged as failed

1. choose random

Ring
1. Choose random
2. Ack lost/too slow
1. choose random
2. ack lost/ too slow
3. ping-req to k random nodes
1. Choose random
2. Ack lost/too slow
3. Ping-req to k random nodes
4. Delegated ping
1. Choose random
2. Ack lost/too slow
3. Ping-req to k random nodes
4. Delegated ping
5. Successful ack
1. choose random
2. ack lost/too slow
3. ping-req to k random nodes
4. delegated ping
5. successful ack
6. ack to initial
Lifeguard Failure Detector

- based on SWIM, developed by Hashicorp
- memberlist implementation
- extensions to the SWIM protocol
- drastically reduces the amount of false-positives

Table 4: Aggregated false positive results for all experiments where \( \alpha = 5 \) and \( \beta = 6 \). For each combination of Lifeguard Components, FP Events is the total number of false positive events, and FP- Events is the number of false positive events at healthy nodes. FP % L0 and FP- % L0 give the same results as the percentage of their respective values for L0.
Dissemination

Pulse (1995)
Dissemination

How to communicate changes in the cluster?

• members joining
• members leaving
• members failing
Dissemination strategies: multicast

Camille Fournier @skamille

Just saw the phrase "multicast support" and a chill went down my spine

6:56 PM - Mar 16, 2018

❤️ 39 🤖 See Camille Fournier's other Tweets
Dissemination strategies: multicast

Quorum of One  
@postwait

The almost uniform inability to support multicast on today's networks is such a humiliating defeat for distributed systems. I just had this conversation today @uber.

8:57 PM - Mar 16, 2018

❤️ 3  👥 See Quorum of One's other Tweets
Dissemination strategies: multicast

- hardware / IP / UDP multicast: not readily (or willingly) enabled in data centres
- even if we had multicast support we'd still have quite a bit of work to do

24 X. Défago, A. Schiper, P. Urbán: Total Order Broadcast and Multicast Algorithms: Taxonomy and Survey
Dissemination strategies: gossip protocols

• based on the research done in the P2P days ¹⁰ ¹¹ ¹⁵ ²⁶

¹¹ S. Ranganathan et al: Gossip-Style Failure Detection and Distributed Consensus for Scalable Heterogeneous Clusters (2000)
¹⁵ I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, H. Balakrishnan: Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications
²⁶ P. Rama, A. D. George, M. Radlinski, R. SubramaniyanL GEMS: Gossip-Enabled Monitoring Service for Heterogeneous Distributed Systems
Gossip styles

- gossip to one node at random\(^\text{10}\)

\(^\text{10}\) van Renesse et al: A gossip-style failure detection service (1998)
Gossip styles

- gossip to one node at random \(^{10}\)
- **round-robin**, binary round-robin, round-robin with sequence check \(^{11}\)

---

\(^{10}\) van Renesse et al: A gossip-style failure detection service (1998)

\(^{11}\) S. Ranganathan et al: Gossip-Style Failure Detection and Distributed Consensus for Scalable Heterogeneous Clusters (2000)

@elmanu - https://manuel.bernhardt.io
Gossip styles

- gossip to one node at random
- round-robin, binary round-robin, round-robin with sequence check

---

11 S. Ranganathan et al: Gossip-Style Failure Detection and Distributed Consensus for Scalable Heterogeneous Clusters (2000)
Gossip styles

- gossip to one node at random \(^{10}\)
- round-robin, binary round-robin, round-robin with sequence check \(^{11}\)
- piggy-back on another protocol (e.g. on a failure detector \(^6\)): also called *infection-style* gossip

---

\(^{10}\) van Renesse et al: A gossip-style failure detection service (1998)

\(^{11}\) S. Ranganathan et al: Gossip-Style Failure Detection and Distributed Consensus for Scalable Heterogeneous Clusters (2000)

What do you even gossip about
What do you even gossip about
What do you even gossip about

\[
\begin{array}{c|c|c}
A & 0 & A \\
B & 1 & 3 \\
C & 3 & B \\
\end{array} + \begin{array}{c|c|c}
A & 0 & A \\
B & 0 & 1 \\
C & 1 & 0 \\
\end{array} = \begin{array}{c|c|c}
A & 0 & A \\
B & 0 & 0 \\
C & 1 & 1 \\
\end{array}
\]
Example of gossip optimizations

- **Akka Cluster**: gossip with a higher probability to nodes that have not already seen a gossip

- **Akka Cluster**: speeds up gossip (3x) when less than half of the members have seen the latest gossip

- **Lifeguard**: anti-entropy mechanism based on nodes doing a full sync with a node at random (helps to speed up convergence after a network partition)
Consensus
A momentary lapse of reason (1987)
Designing consensus protocols

He who fights with monsters should look to it that he himself does not become a monster. And if you gaze long into an abyss, the abyss also gazes into you.

— Friedrich Nietzsche, Beyond Good and Evil, Aphorism 146
Distributed Systems Consensus timeline:

- 1989: consensus is Paxos.
- 2013: consensus is Raft.
- 2018: consensus is that computers are terrible.

10:01 AM - Jul 9, 2018

1,588 people are talking about this
Impossibility result - group membership

Group membership with a single group is impossible when there are nodes that are suspected of having failed.

---

It would be unwise to make membership-related **decisions** while there are processes suspected of having crashed.
Reaching consensus: time

- **Lamport Clocks** \(^{12}\): how do you order events in a distributed system

\(^{12}\) L. Lamport: Time, Clocks, and the Ordering of Events in a Distributed System (1978)
Reaching consensus: time

- **Lamport Clocks**: how do you order events in a distributed system

- **Vector Clocks**: how do you order events in a distributed system and flag *concurrent* event

---

Reaching consensus: time

- **Lamport Clocks**: how do you order events in a distributed system
- **Vector Clocks**: how do you order events in a distributed system and flag concurrent events
- **Version Vectors**\(^{14}\)\(^{15}\) and **Dotted Version Vectors**\(^{16}\): very similar but semantics primarily concerned with versioning and conflict detection in replicas

\(^{14}\) D.S. Parker: Detection of mutual inconsistency in distributed systems (1983)
\(^{15}\) https://haslab.wordpress.com/2011/07/08/version-vectors-are-not-vector-clocks/
\(^{16}\) N. Preguiça: Dotted Version Vectors: Efficient Causality Tracking for Distributed Key-Value Stores (2012)
Reaching consensus: time

private[cluster] final case class Gossip(
  members: immutable.SortedSet[Member],
  overview: GossipOverview = GossipOverview(),
  version: VectorClock = VectorClock(),
  tombstones: Map[UniqueAddress, Gossip.Timestamp] = Map.empty)
Reaching consensus: replicated state machines

Any sufficiently complicated model class contains an ad-hoc, informally-specified, bug-ridden, slow implementation of half a state machine

— Pete Forde
Reaching consensus: replicated state machines

This method allows one to implement any desired form of multiprocess synchronization in a distributed system

— Leslie Lamport ¹²

¹² L. Lamport: Time, Clocks, and the Ordering of Events in a Distributed System (1978)
Reaching consensus: protocols

- **fault-tolerant** distributed systems
- how do multiple servers agree on a value?
- Paxos \(^{19, 20}\), Raft \(^{21}\), Flexible Paxos \(^{22}\), CASPaxos \(^{23}\)

---

\(^{20}\) L. Lamport: Paxos made simple (2001)
\(^{22}\) H. Howard, D. Malkhi, A. Spiegelman: Flexible Paxos: Quorum intersection revisited (2016)
\(^{23}\) D. Rystsov: CASPaxos: Replicated State Machines without logs (2018)
Reaching consensus: CRDTs

- Conflict-Free Replicated Data types

- strong eventual consistency

- two families:
  - CmRDTs (commutativity of operations)
  - CvRDTs (convergence of state - merge function)

---

17 F.B. Schneider: Implementing Fault Tolerant Services Using the State Machine Approach: A Tutorial (1990)
Reaching consensus: conventions

• reaching a decision by not transmitting any information

• example: Akka Cluster leader designation

/**
 * INTERNAL API
 * Orders the members by their address except that members with status
 * Joining, Exiting and Down are ordered last (in that order).
 */
private[cluster] val leaderStatusOrdering: Ordering[Member] = ...
In practice: Akka Cluster
Akka Cluster

- **failure detector**: φ Adaptive Accrual FD with ping-pong
- **dissemination**: random biased gossip
- **consensus**:
  - leader by convention
  - membership decisions driven by leader (joining, leaving)
Akka Cluster: membership states
Akka Cluster: happy path

@elmanu - https://manuel.bernhardt.io
Akka Cluster: sad path (aka reality)
Akka Cluster: how do you even
Where do we go from here?
Where do we go from here?

• at the end of the day, cluster nodes are deployed on physical machines

• nodes of clusters that serve different purposes (Akka, Kafka, Cassandra, ...) co-located on a physical machine
Where do we go from here?

• at the end of the day, cluster nodes are deployed on physical machines

• nodes of clusters that serve different purposes (Akka, Kafka, Cassandra, ...) co-located on a physical machine

• everybody is asking the same questions: who is out there? are you still here? can you hear me?
Temperature Anomalies by Country
Years 1880 - 2017

Data Source:
NASA GISS, GISTEMP Land-Ocean Temperature Index (LOTI), ERSSTv5, 1200km smoothing
https://data.giss.nasa.gov/gistemp/

@elmanu - h+ps:/ /manuel.bernhardt.io
Failure detectors

- Chandra, Toueg: Unreliable failure detectors for reliable distributed systems (1996)
- https://github.com/hashicorp/memberlist
Impossibility results

- M.J. Fischer, N.A. Lynch, and M.S. Paterson: Impossibility of distributed consensus with one faulty process (1985)
- Chandra, Toueg: Unreliable failure detectors for reliable distributed systems (1996)
Dissemination

• van Renesse et al: A gossip-style failure detection service (1998)

• S. Ranganathan et al: Gossip-Style Failure Detection and Distributed Consensus for Scalable Heterogeneous Clusters (2000)


• X. Défago, A. Schiper, P. Urbán: Total Order Broadcast and Multicast Algorithms: Taxonomy and Survey

• I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, H. Balakrishnan: Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications

@elmu - https://manuel.bernhardt.io
Consensus - time

- L. Lamport: Time, Clocks, and the Ordering of Events in a Distributed System (1978)
- D.S. Parker: Detection of mutual inconsistency in distributed systems (1983)
Consensus - protocols

- L. Lamport: Paxos made simple (2001)
- D. Rystsov: CASPaxos: Replicated State Machines without logs (2018)
Consensus - misc

• M. Shapiro: Conflict-free Replicated Data Types (2011)
Thank you!

- Questions?
- Website: https://manuel.bernhardt.io
- Twitter: @elmanu