Conventional approaches to distributed computing

David Barrett-Lennard

16 Oct 2016

# The fallacies of distributed computing

One of the prime motivations of CEDA is to support distributed data management, despite the fact that networks tend to be unreliable, have high latency, low bandwidth etc. Programmers new to distributed applications invariably make the following false assumptions

1. The network is reliable.
2. Latency is zero.
3. Bandwidth is infinite.
4. The network is secure.
5. Topology doesn't change.
6. There is one administrator.
7. Transport cost is zero.
8. The network is homogeneous.

# Some impossibility results in distributed computing

# Two generals' problem

There are some impossibility results in distributed computing, such as the two general's problem.

It is impossible to  coordinate  simultaneous  action. This doesn't contradict atomic commit protocols like 2PC which allow for coordinating eventual agreement. i.e. under 2PC all nodes eventually agree to commit or abort the transaction (assuming they eventually recover).

Armies want consensus to attack at the same time

Messages may be lost

(but not tampered with)

IMPOSSIBLE

# FLP Theorem

Fischer, Lynch and Patterson 1985. Won the Dijkstra award given to the most influential papers in distributed computing.

The FLP theorem states that in an asynchronous network where messages may be delayed but not lost, there is no consensus algorithm that is guaranteed to terminate in every execution for all starting conditions, if at least one node may fail-stop.

Asynchronous network

No lost messages

At least one node may fail-stop

⇒

no consensus algorithm is guaranteed to terminate in every execution for all starting conditions

No upper bound on time to send/process/respond to messages

process fails by stopping forever

Asynchronous: There is no upper bound on the amount of time processors may take to receive, process and respond to an incoming message. Therefore it is impossible to tell if a processor has failed, or is simply taking a long time to do its processing.

Processors are allowed to fail according to the fail-stop model – this simply means that processors that fail do so by ceasing to work correctly.

# Independent recovery

No independent recovery means that nodes are unavailable even though they haven't failed.

Theorem:

There exists no distributed commit protocol with independent process recovery in the presence of multiple failures.

# CAP theorem

Another impossibility result, the CAP theorem says you can't have all three of Consistency, Availability and Partition Tolerance. Under OT we give up on strong consistency and only require consistency once all operations have been received at all sites (i.e. we allow temporary divergence and only require eventual consistency).

(strong)

**C**onsistency

**A**vailability

**P**artition Tolerance

Pick two

Every non-failing node can respond

Distributed object behaves as though all operations are performed in some total order and each operation looks as though it is completed in a single instant

3PC

OT

2PC

Paxos

# Distributed transactions are evil!

Distributed transactions are slow and error prone. It's easy to find negative comments about them on the Internet

*It comes as a shock to many, but distributed transactions are the bane of high performance and high availability*

*...Yes they’re evil. But are they a necessary evil – that’s the real question. ...I believe they are indeed the worst sort: necessary. We’re all dooooooomed......*

*... at the end of the day, however, it is impossible. Nature finds a way to make your life miserable, and all because of these horrid distributed transactions.*

(some humorous comments on the Internet)

# The dangers of replication

In 1996 Jim Gray, Pat Hellend, Patrick O'Neil, Dennis Shasha published a paper in 1996 on Update-anywhere-anytime transactional replication, showing under quite general assumptions that there is a scalability problem:

1000x increase in deadlocks or reconciliations

10x increase in nodes and traffic

⇒

Their conclusions were somewhat pessimistic:

*...this is a bleak picture, but probably accurate*

*...explains why there are no high-update-traffic replicated databases with globally serialisable transactions*

CEDA avoids certain assumptions made in this paper and allows for merging which is *linear* in the number of nodes and traffic.

# 2 Phase Commit (an atomic commitment protocol)

For distributed transactions (an Atomic Commitment Protocol)

Coordinator

Phase 1

Phase 2

Cohort1

Cohort2

prepare

prepare

vote

vote

Coordinator

Cohort1

Cohort2

outcome

outcome

ack

ack

The protocol assumes that there is stable storage at each node with a write-ahead log, that no node crashes forever, that the data in the write-ahead log is never lost or corrupted in a crash, and that any two nodes can communicate with each other.

In phase 1:
\* The coordinator sends a PREPARE message to all cohorts
\* Each cohort may respond with an ABORT-VOTE, or stably record everything needed to later commit the transaction and respond with a COMMIT-VOTE

The coordinator records the outcome in its log (COMMIT iff all cohorts voted to COMMIT, ABORT otherwise).

In phase 2:
\* The coordinator sends an outcome message (COMMIT-VOTE or ABORT-VOTE) to all cohorts
\* The cohort writes the transaction outcome to its log (after this, it never needs to ask the coordinator for the outcome again if it crashes), and replies with an ACK message.

The coordinator writes an END message to its log, which tells it not to reconstruct in-memory information about the transaction on recovering after a crash.

# CAP under 2PC or Paxos

2PC: If the coordinator and a cohort crash then other cohorts may block indefinitely while holding mutexes and they can't commit or abort.

Paxos: Unavailable if a majority (over half of the nodes) can't communicate.

**C**onsistency

**A**vailability

**P**artition Tolerance


# CAP under 3PC

If the network partitions 3PC can give inconsistent states

**C**onsistency

**A**vailability

**P**artition Tolerance


# Location transparency

There are fundamental differences between in-process and inter-process communication that ruin the whole premise of location transparency.

Process A

Proxy

App1

Process B

Object

In-process

Infallible

Latency = 10ns

Inter-process

Fallible

Latency = 100ms

App2