# Sistemas Distribuidos: ¿Para Qué?

Mariano Guerra @warianoguerra

instadeq.com @instadeq

Erlang Factory Light

Buenos Aires 2017


XXXXX
@peterbourgon
Follow
designing coördination-free distributed systems is one of life's great pleasures. not implementing, mind you. that bit is awful.
RETWEETS
26
LIKES
90
12:14 PM - 29 Nov 2016
6
26
90

# Disclaimer

New boring CRUD app, how can we make it exciting?

DISTRIBUTE

# What could possibly go wrong?

Well ... let's see

```
A = foo()
B = bar(A)
do_something(A, B)
```

## Tracing/Metrics

Dapper, a Large-Scale Distributed Systems Tracing Infrastructure

- Open Tracing/zipkin
- Erlang gh:project-fifo/otters

# Logs

Collection, Centralization, Search, Correlation

ELK, Splunk, flume, fluentd, "hadoop"

## Timeouts, Retries, Exponential Backoff

gh:Netflix/Histrix

### Latency and Fault Tolerance

Stop cascading failures.

Fallbacks and graceful degradation.

Circuit Breakers.

## Timeouts, Retries, Exponential Backoff

gh:Netflix/Histrix

### Realtime Operations

Realtime monitoring and configuration changes.

Watch service and property changes take effect immediately.

## Timeouts, Retries, Exponential Backoff

gh:Netflix/Histrix

### Concurrency

Parallel execution.

## New Kinds of Errors

Leaky Abstractions

Timeouts

Transport Errors

Encoding/Parsing Errors

API Versioning

# Beware of the Distributed Monolith

## Fallacies of Distributed Computing

- The network is reliable.
- Latency is zero.
- Bandwidth is infinite.
- The network is secure.
- Topology doesn't change.
- There is one administrator.
- Transport cost is zero.
- The network is homogeneous.

# Partial Failure

## Gray Failure

When at least one **app** makes the observation

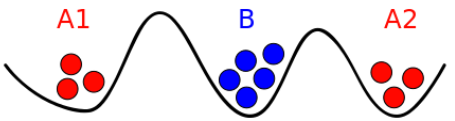
That the system is unhealthy

But the **observer** observes

That the system is healthy.

# Two Generals Problem

Pitfalls and design challenges of attempting to coordinate an action by communicating over an unreliable link.



First computer communication problem to be proved to be **unsolvable**.

## FLP Imposibility Result

In an asynchronous network

## FLP Imposibility Result

In an asynchronous network

Where messages may be delayed but not lost

## FLP Imposibility Result

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

## FLP Imposibility Result

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.

It's about consensus

Deals with ‘faulty nodes’

Nodes that aren’t receiving the messages that are being sent to them are failed (Exempt from having to achieve consensus)

Partitioned node in FLP does not have to achieve consensus,

Since it is considered failed, but the same node in CAP must

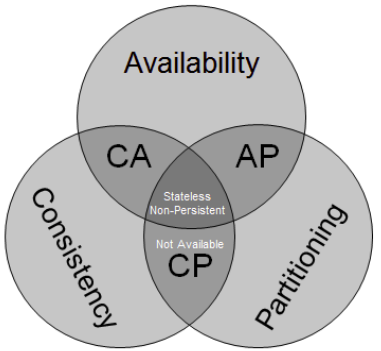
Therefore it’s not possible to say whether a processor has crashed or is simply taking a long time to respond.

The FLP result shows that in an asynchronous setting, where only one processor might crash,

There is no distributed algorithm that solves the consensus problem.

# CAP theorem

# CAP theorem


Availability
CA
AP
Stateless
Non-Persistent
Not Available
CP
Consistency
Partitioning

## Brewer's Conjecture and the Feasibility of Consistent, Available, Partition-Tolerant Web Services

In an asynchronous network

Where messages may be lost

It is impossible to implement a sequentially consistent atomic read/write register

That responds eventually to every request under every pattern of message loss.

It's about serialized atomic objects (registers)

Deals with ‘partitions’

Nodes that aren’t receiving the messages that are being sent to them are only partitioned

A CAP solution requires that any live node be able to correctly serve requests, even if it has not received any messages

# That A, it doesn't mean what you thing it means

Availability refers to a liveness property of an algorithm where every request to a non-failing node must eventually return a valid response.

Not Availability of the system, the system can fail for other reasons.

Don’t Settle For Eventual Consistency

The Limits of the CAP Theorem

## Scalability! But at what COST?

COST of distributed systems: the Configuration that Outperforms a Single Thread

| scalable system | cores | twitter | uk-2007-05 |
|---|---|---|---|
| GraphChi [10] | 2 | 3160s | 6972s |
| Stratosphere [6] | 16 | 2250s | - |
| X-Stream [17] | 16 | 1488s | - |
| Spark [8] | 128 | 857s | 1759s |
| Giraph [8] | 128 | 596s | 1235s |
| GraphLab [8] | 128 | 249s | 833s |
| GraphX [8] | 128 | 419s | 462s |
| Single thread (SSD) | 1 | 300s | 651s |
| Single thread (RAM) | 1 | 275s | - |

**Table 2: Reported elapsed times for 20 PageRank iterations, compared with measured times for single-threaded implementations from SSD and from RAM. GraphChi and X-Stream report times for 5 PageRank iterations, which we multiplied by four.**

| scalable system | cores | twitter | uk-2007-05 |
|---|---|---|---|
| Stratosphere [6] | 16 | 950s | - |
| X-Stream [17] | 16 | 1159s | - |
| Spark [8] | 128 | 1784s | $\geq$ 8000s |
| Giraph [8] | 128 | 200s | $\geq$ 8000s |
| GraphLab [8] | 128 | 242s | 714s |
| GraphX [8] | 128 | 251s | 800s |
| Single thread (SSD) | 1 | 153s | 417s |

**Table 3: Reported elapsed times for label propagation, compared with measured times for single-threaded label propagation from SSD.**

| scalable system | cores | twitter | uk-2007-05 |
|---|---|---|---|
| GraphLab | 128 | 249s | 833s |
| GraphX | 128 | 419s | 462s |
| Vertex order (SSD) | 1 | 300s | 651s |
| Vertex order (RAM) | 1 | 275s | - |
| Hilbert order (SSD) | 1 | 242s | 256s |
| Hilbert order (RAM) | 1 | 110s | - |

**Table 4: Reported elapsed times for 20 PageRank iterations, compared with measured times for single-threaded implementations from SSD and from RAM. The single-threaded times use identical algorithms, but with different edge orders.**

## Scalability! But at what COST?

Many published systems have unbounded COST

(They never outperform the best single threaded application)

Others are orders of magnitude slower even when using hundreds of cores.


THIS IS FINE.

Lord, grant me the strength to accept systems that can run in one node/process,

Lord, grant me the strength to accept systems that can run in one node/process,

the courage to handle the distributed systems I have to,

Lord, grant me the strength to accept systems that can run in one node/process,

the courage to handle the distributed systems I have to,

and the wisdom to know the difference.

But ...

# Cars are distributed systems

"Analysts predict that the on-board computing power of a normal saloon will increase by 100x from 2016 to 2025, powering ADAS and IVI functions."

"ADAS combines information from the many sensors dotted all over the car, feeding into a large processing unit that makes sense of the data, and makes decisions in real time. These sensors include radar, lidar, ultrasonic and cameras"

The future of automotive is coming faster than you think

## What is this?

"Each core has an integrated **network interface** and a **router**, with each router connected to the four routers around it .... There are algorithms designed to reduce **congestion**, ... but a basic system will have **buffers** and **queues** and will know how busy the **local network** congestion is."

# What is this?

"Each core has an integrated **network interface** and a **router**, with each router connected to the four routers around it .... There are algorithms designed to reduce **congestion**, ... but a basic system will have **buffers** and **queues** and will know how busy the **local network** congestion is."

Your new CPU

Intel Skylake-X: The New Core-to-Core Communication Paradigm

You need at least 2 computers for availability

## Other Distributed Systems

- Data Store*
- Mobile Apps
- Data Processing Pipeline*
- IoT
- Microservices
- Currency*
- Games*

## What to do about those?

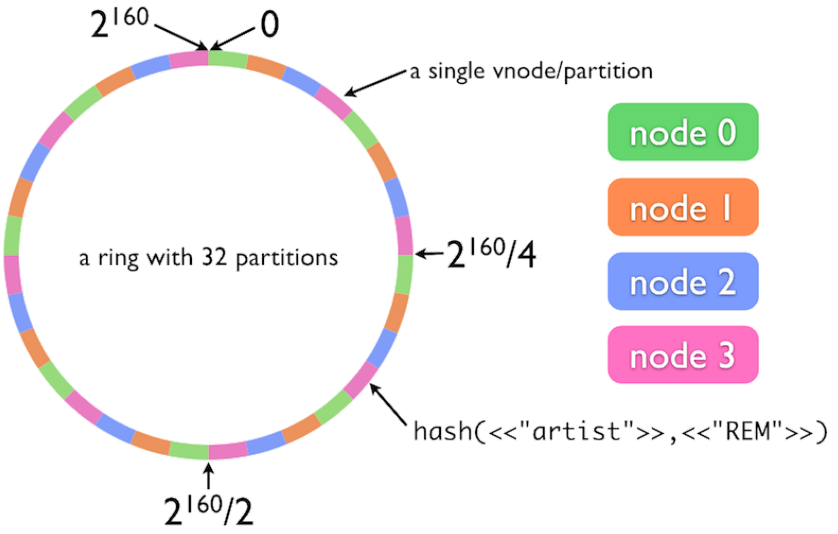
We have to answer at least this questions:

- Who Does This?
- Who Knows This?
- When did This Happen?

And detect problems as soon as posible

# Who Does This?

[Amazon's Dynamo Paper]



# Shameless Plug

[The Little Riak Core Book]

## Distributed Virtual Actors

Orleans - Virtual Actors

Erlang: gh:SpaceTime-IoT/erleans

# Distributed "Transactions"

Sagas

gh:mrallen1/gisla

Spark - Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing

Spark Streaming - Discretized Streams: Fault-Tolerant Streaming Computation at Scale

# Who Knows This?

# Consensus

Paxos

ZAB

Raft (Vis 1, Vis 2)

More

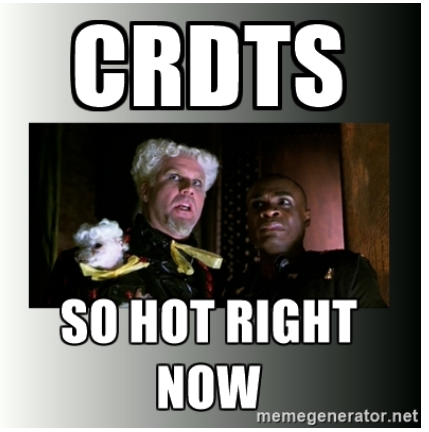
HyParView a membership protocol for reliable gossip-based broadcast

Erlang: lasp/partisan

Epidemic Broadcast Trees

Erlang: plumtree

Convergent and Commutative Replicated Data Types



[A Conflict-Free Replicated JSON Datatype]

Lasp

Lasp is a suite of libraries aimed at providing a comprehensive programming system for planetary scale Elixir and Erlang applications.

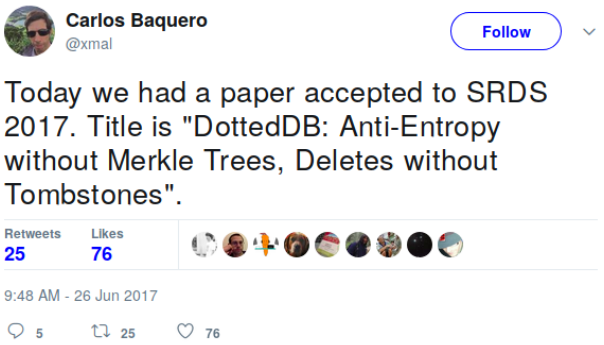
@cmeik

# Merkle Trees

Git, IPFS, riak_core_metadata AAE

How does node 3 gets the values broadcasted while he was down?

Gossip protocols, Epidemic Broadcast and Eventual Consistency in Practice


Carlos Baquero
@xmal
Follow
Today we had a paper accepted to SRDS 2017. Title is "DottedDB: Anti-Entropy without Merkle Trees, Deletes without Tombstones".
Retweets
25
Likes
76
9:48 AM - 26 Jun 2017
5
25
76

Distributed Hash Tables

Bittorrent, IPFS, Erlang global*

Also: Consistent Hashing

gh:jlouis/dht

# When did This Happen?

Time, clocks and the ordering of events in a distributed system

Virtual Time and Global States of Distributed Systems

Dotted Version Vectors: Efficient Causality Tracking for Distributed Key-Value Stores

ACM: Why Logical Clocks are Easy

# Atomic Clocks!

Inside Cloud Spanner and the CAP Theorem

# Avoiding Problems

Simple testing can prevent most critical failures

Early detection of configuration errors to reduce failure damage

Quick Check

Concuerror

TLA+

Lineage-driven Fault Injection

Jepsen

Simian Army

Learn TLA

TLA+ in Practice and Theory Part 1: The Principles of TLA+

```
A = foo()
B = bar(A)
do_something(A, B)
```

Life was simpler, wasn't it?

Let's enjoy single node systems... while we can :)

# Resources

Aphyr's Distributed Systems Class Notes

Designing Data-Intensive Applications

# People

Adrian Colyer @adriancolyer

Aphyr @aphyr

Christopher Meiklejohn @cmeik

Eric Brewer @eric_brewer

Peter Alvaro @palvaro

Peter Bailis @pbailis

# Thanks

Swarm Intelligence: http://ncase.me/fireflies/