ArangoDB

The Computer Science behind a Modern Distributed Database

Chicago / February 20, 2018

Dan Larkin-York

Overview

Topics

- Resilience and Consensus
- Sorting
- Log-structured Merge Trees
- Hybrid Logical Clocks
- Distributed ACID Transactions

Bottom line: You need CompSci to implement a modern data store

Resilience and Consensus

The Problem

A modern data store is **distributed**,

Resilience and Consensus

The Problem

A modern data store is **distributed**, because it needs to **scale out** and/or **be resilient**.

Resilience and Consensus

The Problem

A modern data store is **distributed**, because it needs to **scale out** and/or **be resilient**.

Different parts of the system need to agree on things.

Resilience and Consensus

The Problem

A modern data store is **distributed**, because it needs to **scale out** and/or **be resilient**.

Different parts of the system need to agree on things.

Consensus is the **art** to achieve this **as well as possible** in software.

This is **relatively easy**, if things are good, **but very hard**, if:

Resilience and Consensus

The Problem

A modern data store is **distributed**, because it needs to **scale out** and/or **be resilient**.

Different parts of the system need to agree on things.

Consensus is the **art** to achieve this **as well as possible** in software.

This is **relatively easy**, if things are good, **but very hard**, if:

- the network has **outages**,
- the network has **dropped**, **delayed** or **duplicated** packets,
- **disks fail** (and come back with corrupt data),
- **machines fail** (and come back with old data),
- **racks fail** (and come back with or without data).

Resilience and Consensus

The Problem

A modern data store is **distributed**, because it needs to **scale out** and/or **be resilient**.

Different parts of the system need to agree on things.

Consensus is the **art** to achieve this **as well as possible** in software.

This is **relatively easy**, if things are good, **but very hard**, if:

- the network has **outages**,
- the network has **dropped**, **delayed** or **duplicated** packets,
- **disks fail** (and come back with corrupt data),
- **machines fail** (and come back with old data),
- **racks fail** (and come back with or without data).

(And we have not even talked about malicious attacks and enemy action.)

Paxos and Raft

Traditionally, one uses the **Paxos Consensus Protocol** (1989 . . . 1998).
More recently, **Raft** (2013) has been proposed.

- **Paxos** is **a challenge to understand and to implement efficiently**.
- **Various variants** exist.
- **Raft** is **designed to be understandable**.

Paxos and Raft

Traditionally, one uses the **Paxos Consensus Protocol** (1989 . . . 1998).
More recently, **Raft** (2013) has been proposed.

- **Paxos** is **a challenge to understand and to implement efficiently**.
- **Various variants** exist.
- **Raft** is **designed to be understandable**.

My advice:

First **try to understand Paxos** for some time (do not implement it!), then enjoy **the beauty of Raft**,

Paxos and Raft

Traditionally, one uses the **Paxos Consensus Protocol** (1989 . . . 1998).
More recently, **Raft** (2013) has been proposed.

- **Paxos** is **a challenge to understand and to implement efficiently**.
- **Various variants** exist.
- **Raft** is **designed to be understandable**.

My advice:

First **try to understand Paxos** for some time (do not implement it!), then enjoy **the beauty of Raft**, **but do not implement it either!**

Paxos and Raft

Traditionally, one uses the **Paxos Consensus Protocol** (1989 . . . 1998).
More recently, **Raft** (2013) has been proposed.

- **Paxos** is **a challenge to understand and to implement efficiently**.
- **Various variants** exist.
- **Raft** is **designed to be understandable**.

My advice:

First **try to understand Paxos** for some time (do not implement it!), then enjoy **the beauty of Raft**, **but do not implement it either!**

Use some battle-tested implementation you trust!

Paxos and Raft

Traditionally, one uses the **Paxos Consensus Protocol** (1989 . . . 1998).
More recently, **Raft** (2013) has been proposed.

- **Paxos** is **a challenge to understand and to implement efficiently**.
- **Various variants** exist.
- **Raft** is **designed to be understandable**.

My advice:

First **try to understand Paxos** for some time (do not implement it!), then enjoy **the beauty of Raft**, **but do not implement it either!**

Use some battle-tested implementation you trust!

But most importantly: **DO NOT TRY TO INVENT YOUR OWN!**

Raft in a slide

- An **odd number** of servers each keep a persisted **log of events**.

Raft in a slide

- An **odd number** of servers each keep a persisted **log of events**.
- Everything is **replicated to everybody**.

Raft in a slide

- An **odd number** of servers each keep a persisted **log of events**.
- Everything is **replicated to everybody**.
- They democratically **elect a leader** with absolute majority.

Raft in a slide

- An **odd number** of servers each keep a persisted **log of events**.
- Everything is **replicated to everybody**.
- They democratically **elect a leader** with absolute majority.
- **Only the leader may append** to the replicated log.

Raft in a slide

- An **odd number** of servers each keep a persisted **log of events**.
- Everything is **replicated to everybody**.
- They democratically **elect a leader** with absolute majority.
- **Only the leader may append** to the replicated log.
- An append only counts when **a majority has persisted and confirmed** it.

Raft in a slide

- An **odd number** of servers each keep a persisted **log of events**.
- Everything is **replicated to everybody**.
- They democratically **elect a leader** with absolute majority.
- **Only the leader may append** to the replicated log.
- An append only counts when **a majority has persisted and confirmed** it.
- Very **smart logic** to ensure a **unique leader** and **automatic recovery from failure**.

Raft in a slide

- An **odd number** of servers each keep a persisted **log of events**.
- Everything is **replicated to everybody**.
- They democratically **elect a leader** with absolute majority.
- **Only the leader may append** to the replicated log.
- An append only counts when **a majority has persisted and confirmed** it.
- Very **smart logic** to ensure a **unique leader** and **automatic recovery from failure**.
- It is all **a lot of fun** to get right, but it is **proven to work**.

Raft in a slide

- An **odd number** of servers each keep a persisted **log of events**.
- Everything is **replicated to everybody**.
- They democratically **elect a leader** with absolute majority.
- **Only the leader may append** to the replicated log.
- An append only counts when **a majority has persisted and confirmed** it.
- Very **smart logic** to ensure a **unique leader** and **automatic recovery from failure**.
- It is all **a lot of fun** to get right, but it is **proven to work**.
- One puts a **key/value store** on top, the log contains the **changes**.

Demo

`http://raft.github.io/raftscope/index.html`

(by Diego Ongaro)

Sorting

The Problem

Data stores need **indexes**. In practice, we need to **sort things**.

Sorting

The Problem

Data stores need **indexes**. In practice, we need to **sort things**.

Most published algorithms are **rubbish** on **modern hardware**.

Sorting

The Problem

Data stores need **indexes**. In practice, we need to **sort things**.

Most published algorithms are **rubbish** on **modern hardware**.

The problem is **no longer** the **comparison computations** but the **data movement**.

Sorting

The Problem

Data stores need **indexes**. In practice, we need to **sort things**.

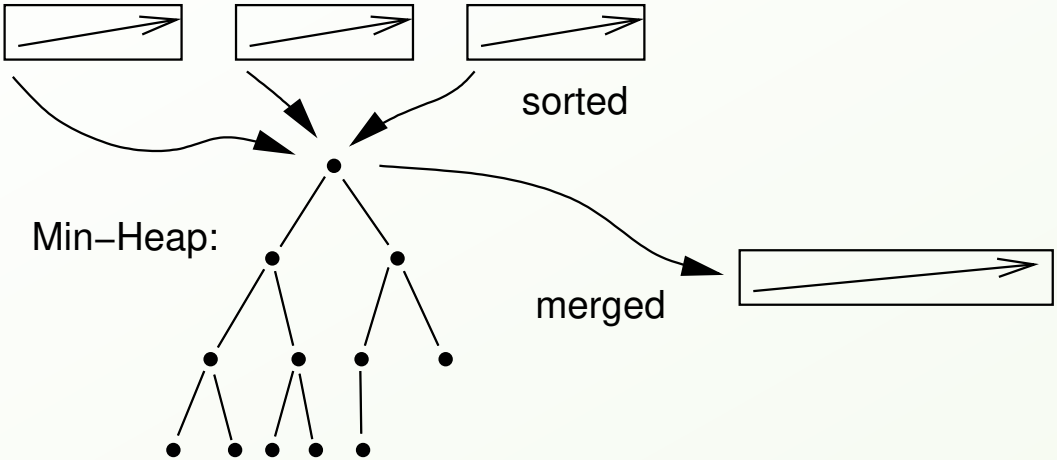
Most published algorithms are **rubbish** on **modern hardware**.

The problem is **no longer** the **comparison computations** but the **data movement**.

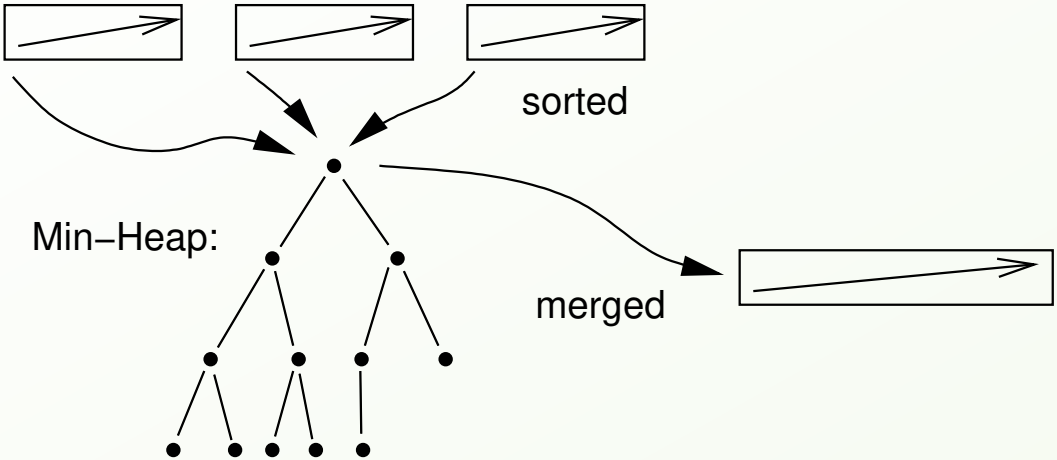
Since 1983 and the Apple IIe,

- compute power in one core has increased by about $\times$**20000**
- and now we have 32 cores in some CPUs
- a single memory access only by about $\times$**40**
- this means **computation** has outpaced **memory access** by $\times$16000!

Idea for a parallel sorting algorithm: Merge Sort



Idea for a parallel sorting algorithm: Merge Sort



Nearly all comparisons hit the L2 cache!

Log structured merge trees (LSM-trees)

The Problem

People rightfully expect from a data store, that it

- can hold **more data than the available RAM**,
- works well **with SSDs and spinning rust**,
- **allows fast bulk inserts** into large data sets, and
- **provides fast reads** in a hot set that fits into RAM.

Log structured merge trees (LSM-trees)

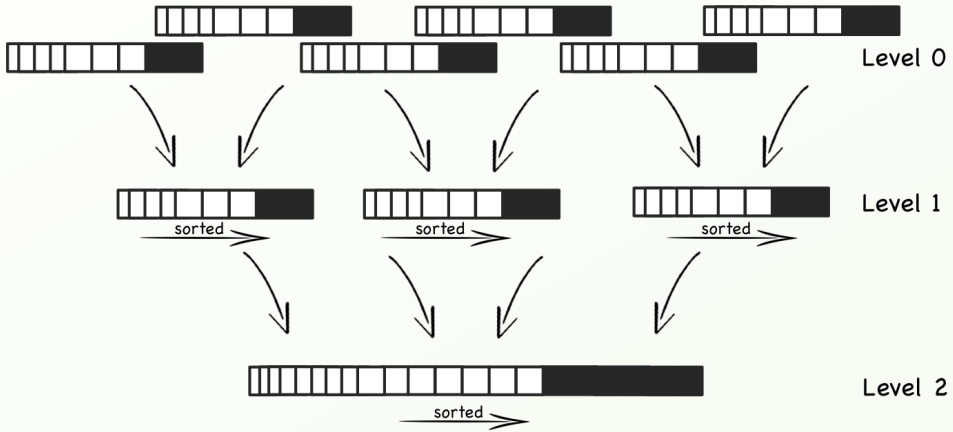
The Problem

People rightfully expect from a data store, that it

- can hold **more data than the available RAM**,
- works well **with SSDs and spinning rust**,
- **allows fast bulk inserts** into large data sets, and
- **provides fast reads** in a hot set that fits into RAM.

Traditional B-tree based structures **often fail to deliver** with the last 2.

Log structured merge trees (LSM-trees)



Compaction continues creating fewer, larger and larger files

(Source: `http://www.benstopford.com/2015/02/14/log-structured-merge-trees/`, Author: Ben Stopford, License: Creative Commons)

Log structured merge trees (LSM-trees)

LSM-trees — summary

- writes **first go into memtables**,
- all files are **sorted** and **immutable**,
- **compaction** happens in the background,
- efficient **merge sort** can be used,
- all writes use **sequential I/O**,
- **Bloom filters** or **Cuckoo filters** for fast negatives,
- $\Longrightarrow$ **good write throughput** and **reasonable read performance**,
- used in ArangoDB, BigTable, Cassandra, FaunaDB, HBase, InfluxDB, LevelDB, MarkLogic, MongoDB, MySQL, RocksDB, SQLite4, WiredTiger, etc.

Hybrid Logical Clocks (HLC)

The Problem

Clocks in different nodes of distributed systems **are not in sync**.

Hybrid Logical Clocks (HLC)

The Problem

Clocks in different nodes of distributed systems **are not in sync**.

- general relativity poses **fundamental obstructions** to synchronicity,
- in practice, **clock skew happens**,
- Google can use **atomic clocks**,
- even with **NTP (network time protocol)** we have to live with $\approx 20ms$.

Hybrid Logical Clocks (HLC)

The Problem

Clocks in different nodes of distributed systems **are not in sync**.

- general relativity poses **fundamental obstructions** to synchronicity,
- in practice, **clock skew happens**,
- Google can use **atomic clocks**,
- even with **NTP (network time protocol)** we have to live with $\approx 20ms$.

Therefore, we **cannot compare time stamps from different nodes!**

Hybrid Logical Clocks (HLC)

The Problem

Clocks in different nodes of distributed systems **are not in sync**.

- general relativity poses **fundamental obstructions** to synchronicity,
- in practice, **clock skew happens**,
- Google can use **atomic clocks**,
- even with **NTP (network time protocol)** we have to live with $\approx 20ms$.

Therefore, we **cannot compare time stamps from different nodes!**

Why would this help?

- establish **"happened after"** relationship between events,
- e.g. for **conflict resolution**, **log sorting**, **detecting network delays**,
- **time to live** could be implemented easily.

Hybrid Logical Clocks (HLC)

The Idea

Every computer has a **local clock**, and we use **NTP** to synchronize.

Hybrid Logical Clocks (HLC)

The Idea

Every computer has a **local clock**, and we use **NTP** to synchronize.
If two events **on different machines** are **linked by causality**, the **cause** should have a smaller time stamp than the **effect**.

Hybrid Logical Clocks (HLC)

The Idea

Every computer has a **local clock**, and we use **NTP** to synchronize.
If two events **on different machines** are **linked by causality**, the **cause** should have a smaller time stamp than the **effect**.

$$\textbf{causality} \iff \textbf{a message is sent}$$

Send a time stamp **with every message**. The HLC always returns a value

$$> \max(\text{local clock}, \text{largest time stamp ever seen}).$$

Hybrid Logical Clocks (HLC)

The Idea

Every computer has a **local clock**, and we use **NTP** to synchronize.
If two events **on different machines** are **linked by causality**, the **cause** should have a smaller time stamp than the **effect**.

$$\textbf{causality} \iff \textbf{a message is sent}$$

Send a time stamp **with every message**. The HLC always returns a value

$$> \max(\text{local clock}, \text{largest time stamp ever seen}).$$

Causality is preserved, time can **"catch up"** with logical time eventually.

`http://muratbuffalo.blogspot.com.es/2014/07/hybrid-logical-clocks.html`

Distributed ACID Transactions

Atomic — either happens **in its entirety** or **not at all**

Consistent — reading **sees a consistent state**, writing **preserves consistency**

Isolated — concurrent transactions **do not see each other**

Durable — committed writes are **preserved after shutdown and crashes**

Distributed ACID Transactions

Atomic — either happens **in its entirety** or **not at all**

Consistent — reading **sees a consistent state**, writing **preserves consistency**

Isolated — concurrent transactions **do not see each other**

Durable — committed writes are **preserved after shutdown and crashes**

(All relatively doable when transactions happen one after another!)

Distributed ACID Transactions

The Problem

In a distributed system:

- How to make sure, that all nodes **agree** on whether the transaction has happened? **(Atomicity)**

Distributed ACID Transactions

The Problem

In a distributed system:

- How to make sure, that all nodes **agree** on whether the transaction has happened? **(Atomicity)**
- How to create a **consistent snapshot** across nodes? **(Consistency)**

Distributed ACID Transactions

The Problem

In a distributed system:

- How to make sure, that all nodes **agree** on whether the transaction has happened? **(Atomicity)**
- How to create a **consistent snapshot** across nodes? **(Consistency)**
- How to **hide ongoing activities** until commit? **(Isolation)**

Distributed ACID Transactions

The Problem

In a distributed system:

- How to make sure, that all nodes **agree** on whether the transaction has happened? **(Atomicity)**
- How to create a **consistent snapshot** across nodes? **(Consistency)**
- How to **hide ongoing activities** until commit? **(Isolation)**
- How to handle **lost nodes**? **(Durability)**

Distributed ACID Transactions

The Problem

In a distributed system:

- How to make sure, that all nodes **agree** on whether the transaction has happened? **(Atomicity)**
- How to create a **consistent snapshot** across nodes? **(Consistency)**
- How to **hide ongoing activities** until commit? **(Isolation)**
- How to handle **lost nodes**? **(Durability)**

We have to take **replication**, **resilience** and **failover** into account.

Distributed ACID Transactions

WITHOUT

Distributed databases without ACID transactions:
ArangoDB, BigTable, Couchbase, Datastax, Dynamo, Elastic, HBase, MongoDB, RethinkDB, Riak, and lots more . . .

WITH

Distributed databases with ACID transactions:
CockroachDB, FaunaDB, ~~FoundationDB~~, MarkLogic, Spanner

Distributed ACID Transactions

WITHOUT

Distributed databases without ACID transactions:
ArangoDB, BigTable, Couchbase, Datastax, Dynamo, Elastic, HBase, MongoDB, RethinkDB, Riak, and lots more . . .

WITH

Distributed databases with ACID transactions:
CockroachDB, FaunaDB, ~~FoundationDB~~, MarkLogic, Spanner

$\Longrightarrow$ **Very few distributed engines promise ACID, because this is hard!**

Distributed ACID Transactions

Basic Idea

- Use **Multi Version Concurrency Control (MVCC)**, i.e. multiple revisions of a data item are kept.

Distributed ACID Transactions

Basic Idea

- Use **Multi Version Concurrency Control (MVCC)**, i.e. multiple revisions of a data item are kept.
- Do **writes** and **replication** decentrally and distributed, **without them becoming visible from other transactions.**

Distributed ACID Transactions

Basic Idea

- Use **Multi Version Concurrency Control (MVCC)**, i.e. multiple revisions of a data item are kept.
- Do **writes** and **replication** decentrally and distributed, **without them becoming visible from other transactions.**
- Then have **some** place, where there is a **switch**, which decides **when the transaction becomes visible**.

Distributed ACID Transactions

Basic Idea

- Use **Multi Version Concurrency Control (MVCC)**, i.e. multiple revisions of a data item are kept.
- Do **writes** and **replication** decentrally and distributed, **without them becoming visible from other transactions.**
- Then have **some** place, where there is a **switch**, which decides **when the transaction becomes visible**.
- These “switches” need to
  - be **persisted** somewhere **(durability)**,
  - **scale out (no bottleneck for commit/abort)**,
  - be **replicated (no single point of failure)**,
  - be **resilient** in case of fail-over **(fault-tolerance)**.

Distributed ACID Transactions

Basic Idea

- Use **Multi Version Concurrency Control (MVCC)**, i.e. multiple revisions of a data item are kept.
- Do **writes** and **replication** decentrally and distributed, **without them becoming visible from other transactions.**
- Then have **some** place, where there is a **switch**, which decides **when the transaction becomes visible**.
- These "switches" need to
  - be **persisted** somewhere **(durability)**,
  - **scale out (no bottleneck for commit/abort)**,
  - be **replicated (no single point of failure)**,
  - be **resilient** in case of fail-over **(fault-tolerance)**.
- **Transaction visibility** needs to be implemented (MVCC), so comparing time stamps play a crucial role.

Thank you!

Further questions?

- Follow us on twitter: `@arangodb`
- Join our slack: `slack.arangodb.com`
- Download and documentation: `https://arangodb.com`
- Issues and source (Star us!):
  `https://github.com/arangodb/arangodb`
- Info and slides:
  `https://arangodb.com/speakers/daniel-larkin-york`

Links

http://the-paper-trail.org/blog/consensus-protocols-paxos

https://raft.github.io

https://en.wikipedia.org/wiki/Merge_sort

http://www.benstopford.com/2015/02/14/log-structured-merge-trees/

http://muratbuffalo.blogspot.com.es/2014/07/hybrid-logical-clocks.html

https://research.google.com/archive/spanner.html

https://www.cockroachlabs.com/docs/cockroachdb-architecture.html

https://www.arangodb.com

http://mesos.apache.org