ArangoDB

Deep dive into the native multi-model database ArangoDB

Percona Live 2016, Santa Clara, 20 April 2016

Frank Celler

Overview

ArangoDB is a **multi-model** Database

Features

- is a document store, a key/value store **and** a graph database,
- offers convenient queries (via HTTP/REST and AQL),
- including joins between different collections,
- and graph queries,
- with configurable consistency guarantees using transactions.

ArangoDB is a **multi-model** Database

Features

- is a document store, a key/value store **and** a graph database,
- offers convenient queries (via HTTP/REST and AQL),
- including joins between different collections,
- and graph queries,
- with configurable consistency guarantees using transactions.

$\Longrightarrow$ Allows **polyglot persistence** with multiple instances of a single technology.

ArangoDB is extensible by JavaScript Code

The Foxx Microservice Framework

Allows you to extend the HTTP/REST API by **your own routes**, which you implement in JavaScript running on the database server, with direct access to the C++ DB engine.

ArangoDB is extensible by JavaScript Code

The Foxx Microservice Framework

Allows you to extend the HTTP/REST API by **your own routes**, which you implement in JavaScript running on the database server, with direct access to the C++ DB engine.

Unprecedented possibilities for data centric services:

- custom-made complex queries or authorizations
- schema-validation
- push feeds, etc.

ArangoDB is a Data Center Operating System App

These days, computing clusters run Data Center Operating Systems.

ArangoDB is a Data Center Operating System App

These days, computing clusters run Data Center Operating Systems.

Idea

Distributed applications can be deployed as easily as one installs a mobile app on a phone.

ArangoDB is a Data Center Operating System App

These days, computing clusters run Data Center Operating Systems.

Idea

Distributed applications can be deployed as easily as one installs a mobile app on a phone.

- Cluster resource management is **automatic**.
- This leads to significantly better **resource utilization**.
- Fault tolerance, self-healing and automatic failover is **guaranteed**.

Details

The Multi-Model Approach

Multi-model database

A multi-model database combines a document store with a graph database and is at the same time a key/value store,
with a common query language for all three data models.

The Multi-Model Approach

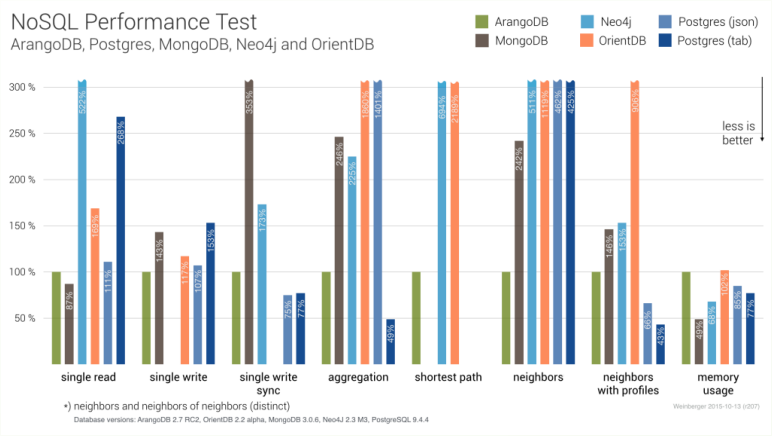
Multi-model database

A multi-model database combines a document store with a graph database and is at the same time a key/value store,
with a common query language for all three data models.

Important:

- is able to compete with specialised products on their turf
- allows for polyglot persistence using a single database technology
- In a microservice architecture, there will be several **different** deployments.

ArangoDB performance



https://www.arangodb.com/2015/10/benchmark-postgresql-mongodb-arangodb/

Why is multi-model possible at all?

Document stores and key/value stores

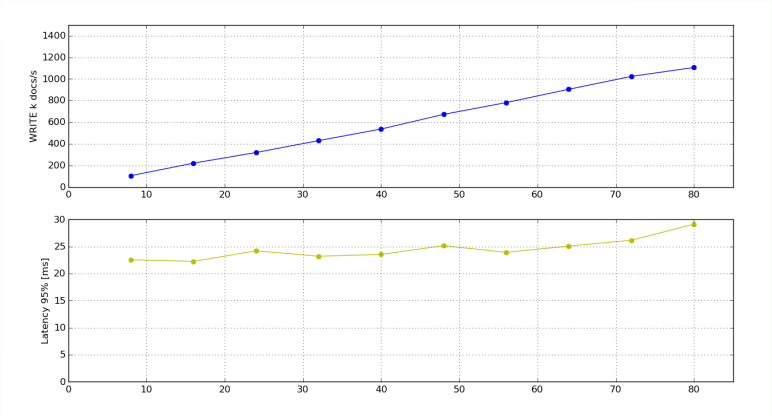
Document stores: have primary key, are key/value stores.

Without using secondary indexes, performance is nearly as good as with opaque data instead of JSON.

Good horizontal scalability can be achieved for key lookups.

ArangoDB horizontal scalability

Experiment: Single document writes (1kB / doc) on cluster of sizes 8 to 80 machines (64 to 640 vCPUs), another 4 to 40 load servers, running on AWS.



https://mesosphere.com/blog/2015/11/30/arangodb-benchmark-dcos/

Why is multi-model possible at all?

Document stores and graph databases

Graph database: would like to associate arbitrary data with vertices and edges, so JSON documents are a good choice.

- A good edge index, giving fast access to neighbours.
  This can be a secondary index.
- Graph support in the query language.
- Implementations of graph algorithms in the DB engine.

https://www.arangodb.com/2016/04/index-free-adjacency-hybrid-indexes-graph-databases/

ArangoDB provides (Version 2.8, January 2016)

- Sharding with automatic data distribution,
- easy setup of (asynchronous) replication (cluster and single),
- fault tolerance by automatic failover,
- full integration with Apache Mesos and Mesosphere DC/OS.

ArangoDB provides (Version 2.8, January 2016)

- Sharding with automatic data distribution,
- easy setup of (asynchronous) replication (cluster and single),
- fault tolerance by automatic failover,
- full integration with Apache Mesos and Mesosphere DC/OS.

Work in progress (Version 3.0, RC in April 2016):

- synchronous replication in cluster mode,
- zero administration by a self-repairing and self-balancing cluster architecture.

Resource Management

- Installation should be as easy as possible
- integration into the resource management of data-center
- gives better resource utilisation,
- full integration with Apache Mesos and Mesosphere DC/OS

Resource Management

- Installation should be as easy as possible
- integration into the resource management of data-center
- gives better resource utilisation,
- full integration with Apache Mesos and Mesosphere DC/OS

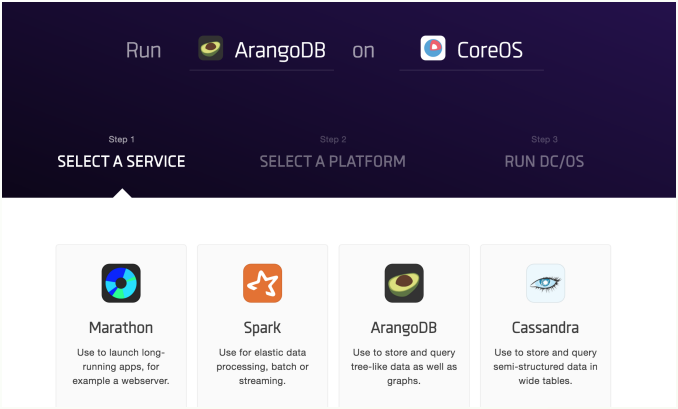
Work in progress

- Mesosphere DC/OS a very mature, Open-Source solution
- later this year integration also for Kubernetes, Docker-Swarm

About Mesosphere's DC/OS

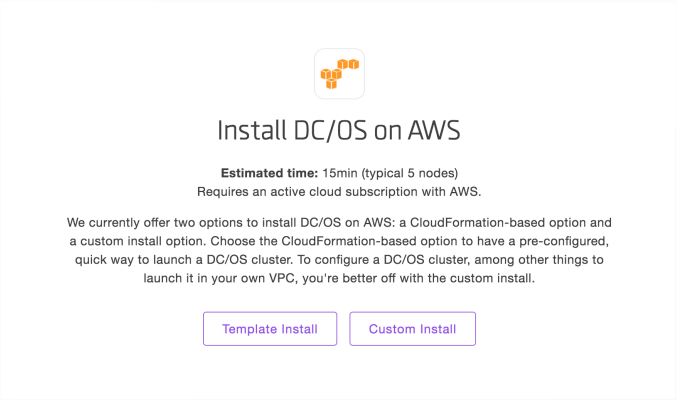
https://dcos.io

Installing Mesosphere's DC/OS



https://dcos.io

Installing Mesosphere's DC/OS



https://dcos.io

Powerful query language

AQL

The built in Arango Query Language allows

- complex, powerful and convenient queries,
- with transaction semantics,
- allowing to do joins,
- AQL is independent of the driver used and
- offers protection against injections by design.

The Foxx Microservice Framework

Allows you to extend the HTTP/REST API by **your own routes**, which you implement in JavaScript running on the database server, with direct access to the C++ DB engine.

The Foxx Microservice Framework

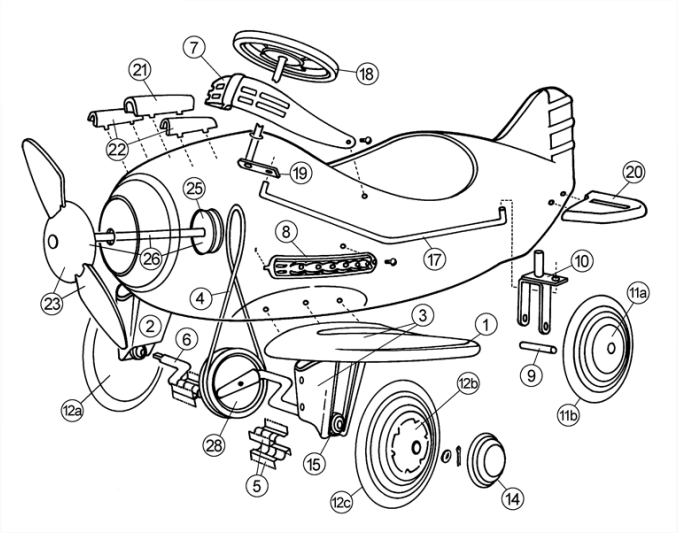
Allows you to extend the HTTP/REST API by **your own routes**, which you implement in JavaScript running on the database server, with direct access to the C++ DB engine.

Unprecedented possibilities for data centric services:

- complex queries or authorizations, schema-validation, push feeds, etc.
- easy deployment via web interface or REST API,
- automatic API description through **Swagger** $\Longrightarrow$ discoverability of services.

Use Cases

Use case: Aircraft fleet management



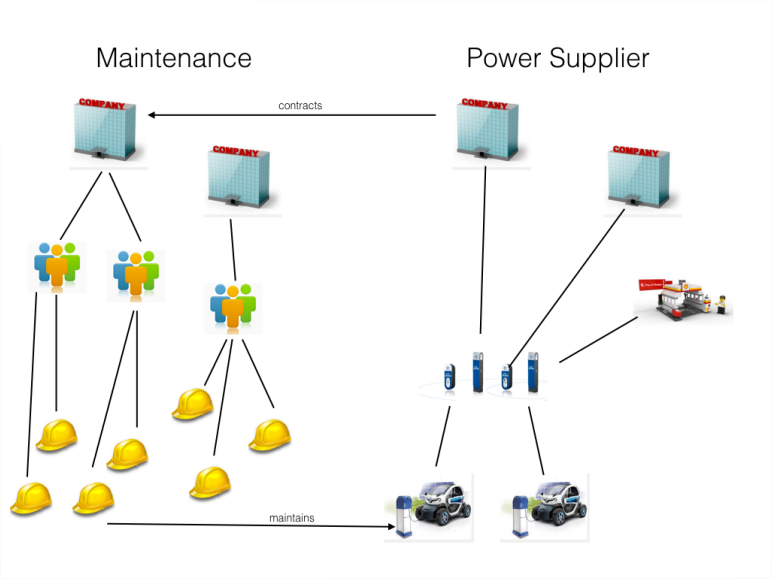
Use case: Aircraft fleet management

One of our customers uses ArangoDB to

- store each part, component, unit or aircraft as a document
- model containment as a graph
- thus can easily find all parts of some component
- keep track of maintenance intervals
- perform queries orthogonal to the graph structure
- thereby getting good efficiency for all needed queries

http://radar.oreilly.com/2015/07/data-modeling-with-multi-model-databases.html

Use case: rights management



Use case: rights management

Right managements in relational model is hard:

- looks like a forest at first
- then exceptions pop-up
- one company sub-contracts another for a special station
- an engineer works for two companies
- some-one needs special permissions when being a proxy
- much easier expressed as graph structure

Use case: e-commerce

Use case: e-commerce

AboutYou uses ArangoDB to

- create channels showing new products
- allow recommendation to friends
- celebrities presenting new fashion
- blog about fashion products
- nightly business analysis
- news stream

https://www.arangodb.com/case-studies/aboutyou-data-driven-personalization-with-arangodb/

Action

First deployment: a simple key/value store

A key/value store

One collection “data”, indexes on “value” (sorted) and “name” (hash).

- Single document requests
- Indexes possible
- Range queries possible

Second deployment: a Microservice as a Foxx app

A Foxx Microservice

Simple TODO app, deployed from app store with web UI.

- REST/JSON API available
- Swagger generates API description automatically

Third deployment: a single server graph database

A Graph Database

Graph “worldCountry” with vertex collection “worldVertex” and edge collection “worldEdges”, links from cities to countries to continents to world.

- Show some graph traversals.
- Show graph viewer.

Fourth deployment: a multi-model application

A multi-model database

Some data from a web shop.

- Show some queries.

AQL Internals

Life of a query

- Text and query parameters come from user

Life of a query

- Text and query parameters come from user
- Parse text, produce **abstract syntax tree (AST)**

Life of a query

- Text and query parameters come from user
- Parse text, produce **abstract syntax tree (AST)**
- Substitute query parameters

Life of a query

- Text and query parameters come from user
- Parse text, produce **abstract syntax tree (AST)**
- Substitute query parameters
- First optimisation: constant expressions, etc.

Life of a query

- Text and query parameters come from user
- Parse text, produce **abstract syntax tree (AST)**
- Substitute query parameters
- First optimisation: constant expressions, etc.
- Translate AST into an **execution plan (EXP)**

Life of a query

- Text and query parameters come from user
- Parse text, produce **abstract syntax tree (AST)**
- Substitute query parameters
- First optimisation: constant expressions, etc.
- Translate AST into an **execution plan (EXP)**
- Optimise one EXP, produce **many, potentially better EXPs**

Life of a query

- Text and query parameters come from user
- Parse text, produce **abstract syntax tree (AST)**
- Substitute query parameters
- First optimisation: constant expressions, etc.
- Translate AST into an **execution plan (EXP)**
- Optimise one EXP, produce **many, potentially better EXPs**
- Reason about **distribution in cluster**

Life of a query

- Text and query parameters come from user
- Parse text, produce **abstract syntax tree (AST)**
- Substitute query parameters
- First optimisation: constant expressions, etc.
- Translate AST into an **execution plan (EXP)**
- Optimise one EXP, produce **many, potentially better EXPs**
- Reason about **distribution in cluster**
- Optimise **distributed** EXPs

Life of a query

- Text and query parameters come from user
- Parse text, produce **abstract syntax tree (AST)**
- Substitute query parameters
- First optimisation: constant expressions, etc.
- Translate AST into an **execution plan (EXP)**
- Optimise one EXP, produce **many, potentially better EXPs**
- Reason about **distribution in cluster**
- Optimise **distributed** EXPs
- Estimate costs for all EXPs, and **sort by ascending cost**

Life of a query

- Text and query parameters come from user
- Parse text, produce **abstract syntax tree (AST)**
- Substitute query parameters
- First optimisation: constant expressions, etc.
- Translate AST into an **execution plan (EXP)**
- Optimise one EXP, produce **many, potentially better EXPs**
- Reason about **distribution in cluster**
- Optimise **distributed** EXPs
- Estimate costs for all EXPs, and **sort by ascending cost**
- Instanciate “cheapest” plan, i.e. set up **execution engine**

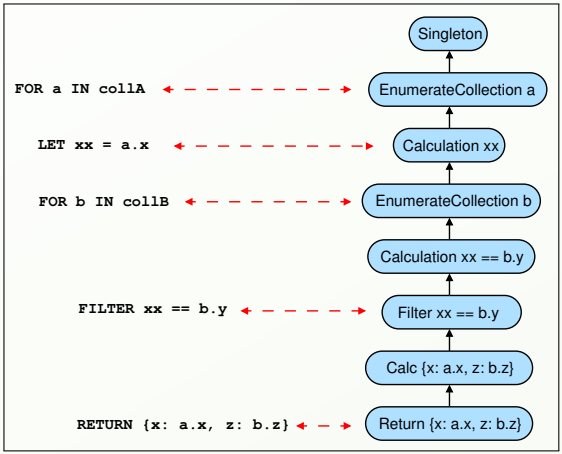
Life of a query

- Text and query parameters come from user
- Parse text, produce **abstract syntax tree (AST)**
- Substitute query parameters
- First optimisation: constant expressions, etc.
- Translate AST into an **execution plan (EXP)**
- Optimise one EXP, produce **many, potentially better EXPs**
- Reason about **distribution in cluster**
- Optimise **distributed** EXPs
- Estimate costs for all EXPs, and **sort by ascending cost**
- Instanciate “cheapest” plan, i.e. set up **execution engine**
- Distribute and link up engines on **different servers**

Life of a query

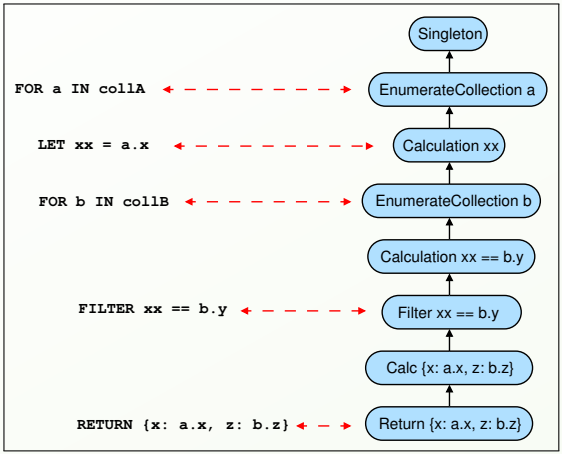
- Text and query parameters come from user
- Parse text, produce **abstract syntax tree (AST)**
- Substitute query parameters
- First optimisation: constant expressions, etc.
- Translate AST into an **execution plan (EXP)**
- Optimise one EXP, produce **many, potentially better EXPs**
- Reason about **distribution in cluster**
- Optimise **distributed** EXPs
- Estimate costs for all EXPs, and **sort by ascending cost**
- Instanciate “cheapest” plan, i.e. set up **execution engine**
- Distribute and link up engines on **different servers**
- Execute plan, provide **cursor API**

Execution plans



Query → EXP

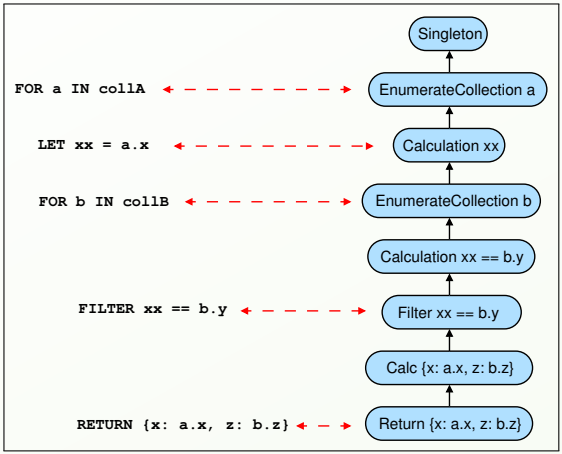
Execution plans



Query → EXP

Black arrows are dependencies

Execution plans

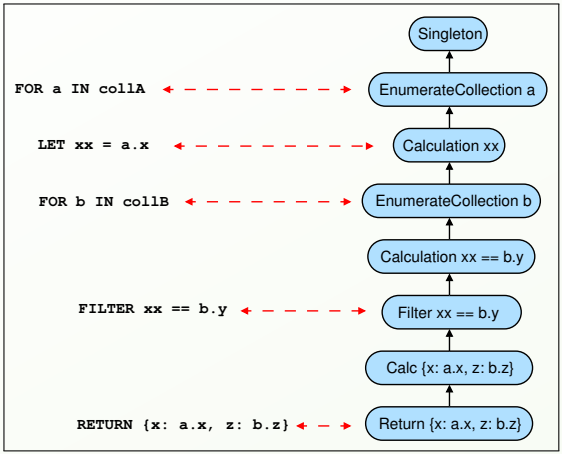


Query → EXP

Black arrows are dependencies

Think of a pipeline

Execution plans



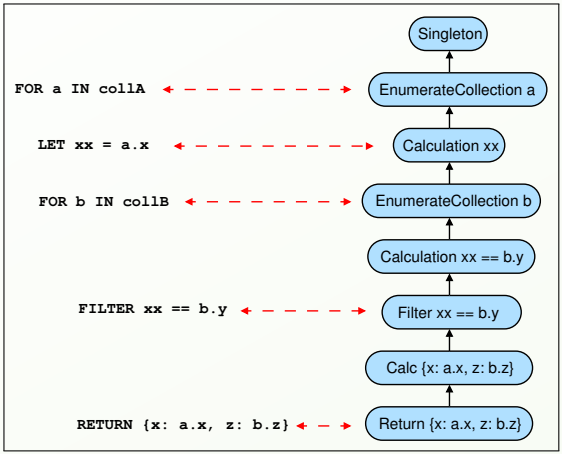
Query → EXP

Black arrows are **dependencies**

Think of a pipeline

Each node provides a **cursor API**

Execution plans



Query → EXP

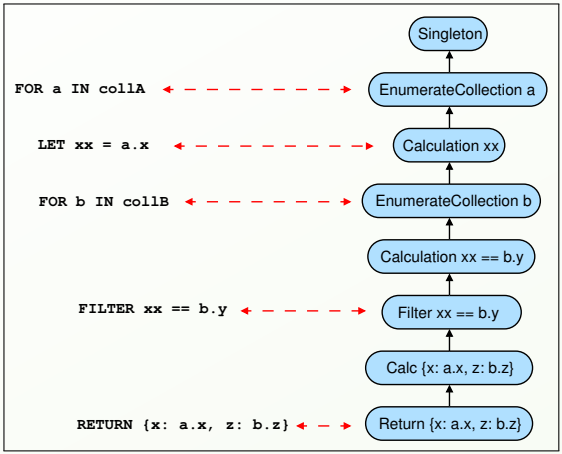
Black arrows are **dependencies**

Think of a pipeline

Each node provides a **cursor API**

Blocks of "Items" travel through the **pipeline**

Execution plans



Query → EXP

Black arrows are **dependencies**

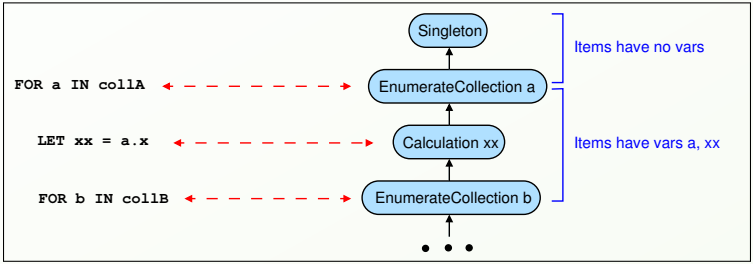
Think of a pipeline

Each node provides a **cursor API**

Blocks of "Items" travel through the **pipeline**

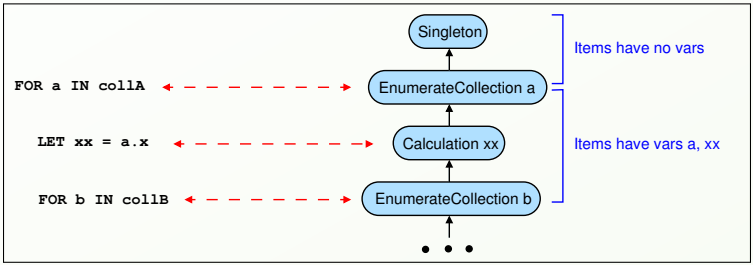
What is an "item"???

Pipeline and items



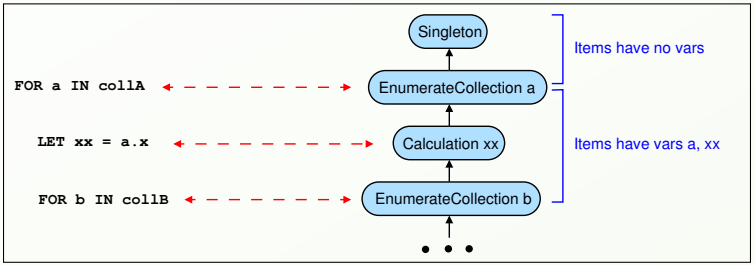
- Items are the **thingies** traveling through the pipeline.

Pipeline and items



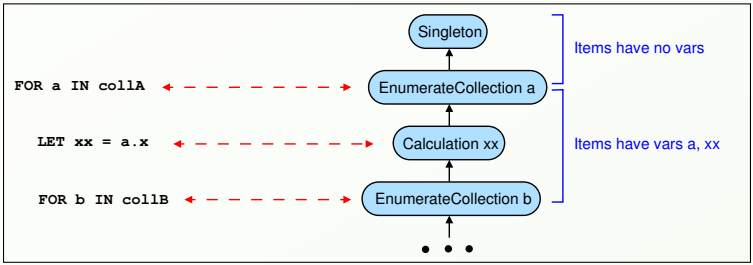
- Items are the **thingies** traveling through the pipeline.
- An item holds **values of those variables in the current frame**

Pipeline and items



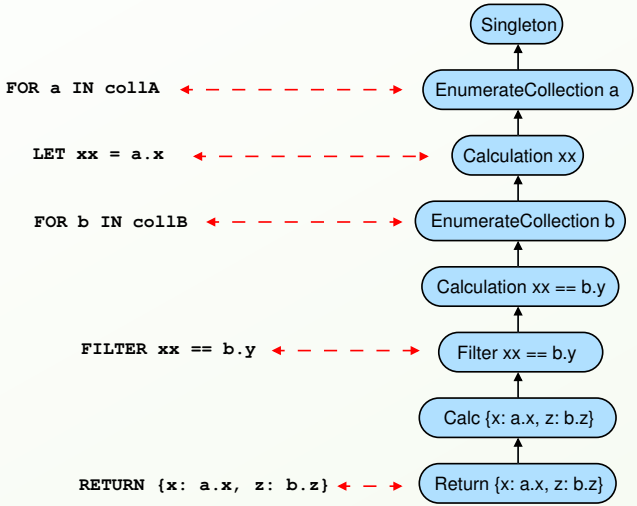
- Items are the **thingies** traveling through the pipeline.
- An item holds **values of those variables in the current frame**
- **Thus:** Items look differently in **different parts of the plan**

Pipeline and items



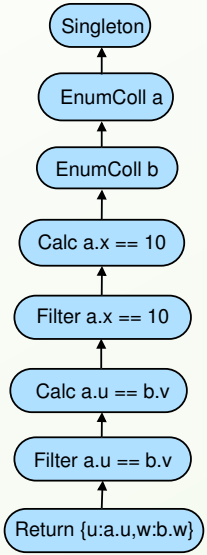
- Items are the **thingies** traveling through the pipeline.
- An item holds **values of those variables in the current frame**
- **Thus:** Items look differently in **different parts of the plan**
- We always deal with blocks of items for **performance reasons**

Execution plans



Move filters up

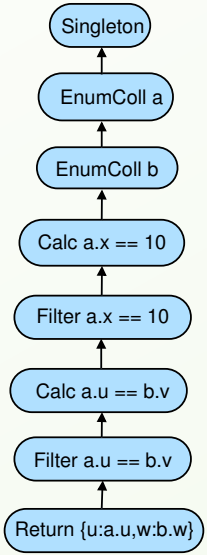
```
FOR a IN collA
  FOR b IN collB
    FILTER a.x == 10
    FILTER a.u == b.v
    RETURN {u:a.u,w:b.w}
```



Move filters up

```
FOR a IN collA
  FOR b IN collB
    FILTER a.x == 10
    FILTER a.u == b.v
    RETURN {u:a.u,w:b.w}
```

The result and behaviour does **not change**, if the first FILTER is pulled out of the inner FOR.

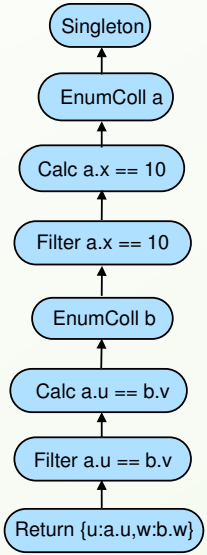


Move filters up

```
FOR a IN collA
  FILTER a.x < 10
  FOR b IN collB
    FILTER a.u == b.v
    RETURN {u:a.u,w:b.w}
```

The result and behaviour does **not change**, if the first FILTER is pulled out of the inner FOR.

However, the number of items traveling in the pipeline **is decreased**.



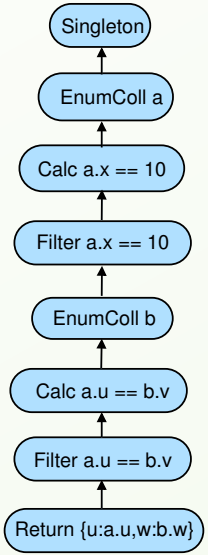
Move filters up

```
FOR a IN collA
  FILTER a.x < 10
  FOR b IN collB
    FILTER a.u == b.v
    RETURN {u:a.u,w:b.w}
```

The result and behaviour does **not change**, if the first FILTER is pulled out of the inner FOR.

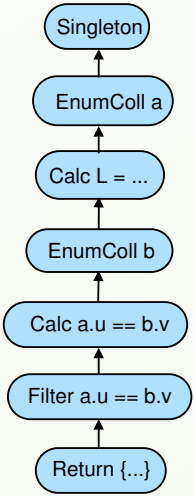
However, the number of items traveling in the pipeline **is decreased**.

Note that the two FOR statements could be interchanged!



Remove unnecessary calculations

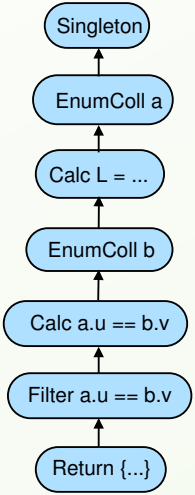
```
FOR a IN collA
  LET L = LENGTH(a.hobbies)
  FOR b IN collB
    FILTER a.u == b.v
    RETURN {h:a.hobbies,w:b.w}
```



Remove unnecessary calculations

```
FOR a IN collA
  LET L = LENGTH(a.hobbies)
  FOR b IN collB
    FILTER a.u == b.v
    RETURN {h:a.hobbies,w:b.w}
```

The Calculation of `L` is unnecessary!



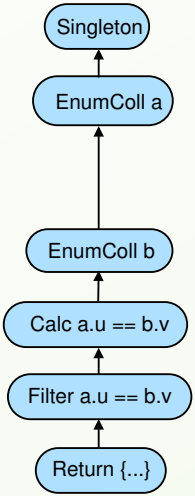
Remove unnecessary calculations

```
FOR a IN collA

  FOR b IN collB
    FILTER a.u == b.v
    RETURN {h:a.hobbies,w:b.w}
```

The Calculation of `L` is unnecessary!

(since it cannot throw an exception).



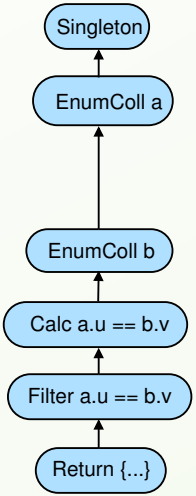
Remove unnecessary calculations

```
FOR a IN collA

  FOR b IN collB
    FILTER a.u == b.v
    RETURN {h:a.hobbies,w:b.w}
```

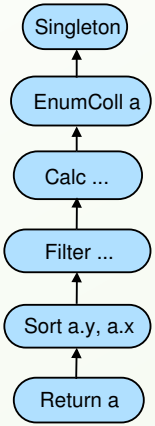
The Calculation of `L` is unnecessary!
(since it cannot throw an exception).

Therefore we can just **leave it out**.



Use index for FILTER and SORT

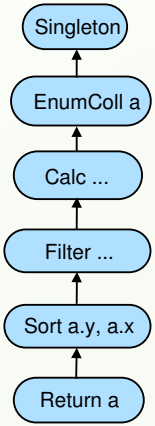
```
FOR a IN collA
  FILTER a.x > 17 &&
         a.x <= 23 &&
         a.y == 10
  SORT a.y, a.x
  RETURN a
```



Use index for FILTER and SORT

```
FOR a IN collA
  FILTER a.x > 17 &&
         a.x <= 23 &&
         a.y == 10
  SORT a.y, a.x
  RETURN a
```

Assume `collA` has a skiplist index on “y” and “x” **(in this order)**,



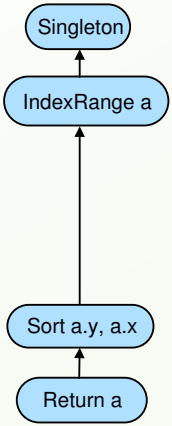
Use index for FILTER and SORT

```
FOR a IN collA
  FILTER a.x > 17 &&
         a.x <= 23 &&
         a.y == 10
  SORT a.y, a.x
  RETURN a
```

Assume `collA` has a skiplist index on “y” and “x” **(in this order)**, then we can read off the half-open **interval** between

`{ y: 10, x: 17 }` and
`{ y: 10, x: 23 }`

from the skiplist index.



Use index for FILTER and SORT

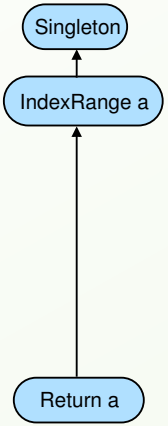
```
FOR a IN collA
  FILTER a.x > 17 &&
         a.x <= 23 &&
         a.y == 10
  SORT a.y, a.x
  RETURN a
```

Assume `collA` has a skiplist index on “y” and “x” **(in this order)**, then we can read off the half-open **interval** between

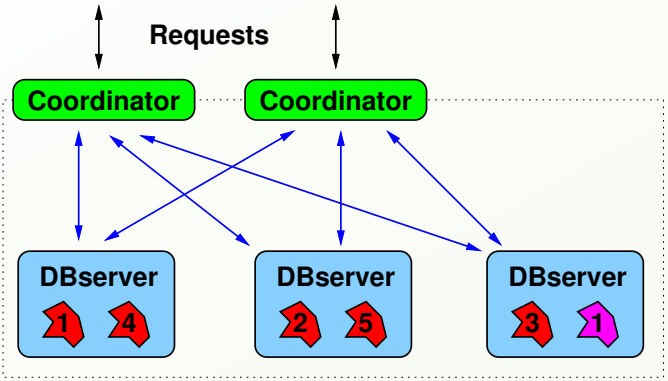
`{ y: 10, x: 17 }` and
`{ y: 10, x: 23 }`

from the skiplist index.

The result will automatically be sorted by `y` and then by `x`.

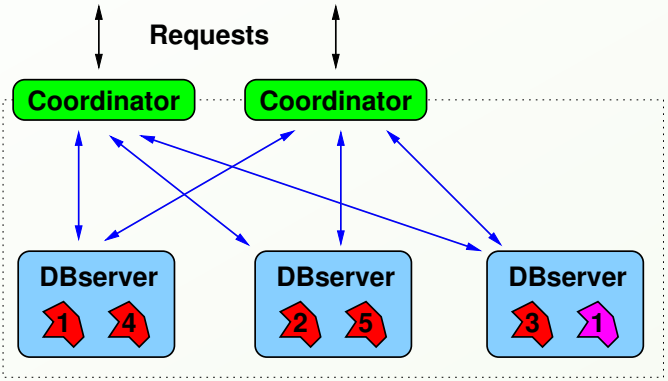


Data distribution in a cluster



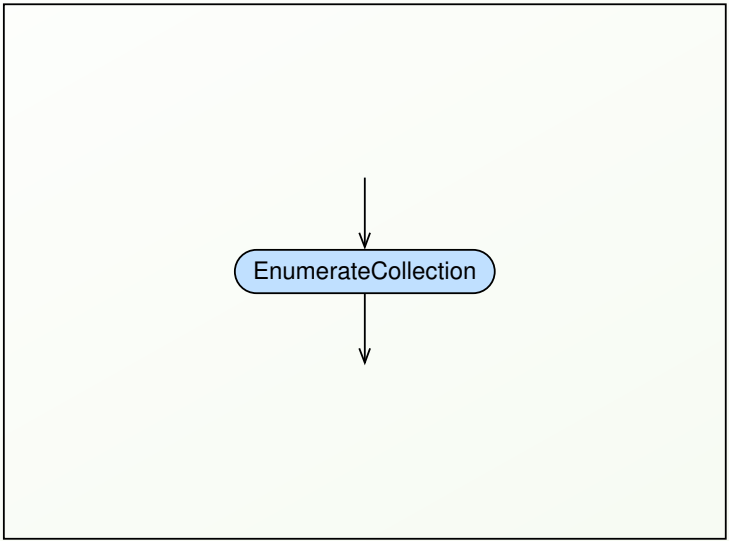
- The shards of a collection are distributed across the DB servers.

Data distribution in a cluster

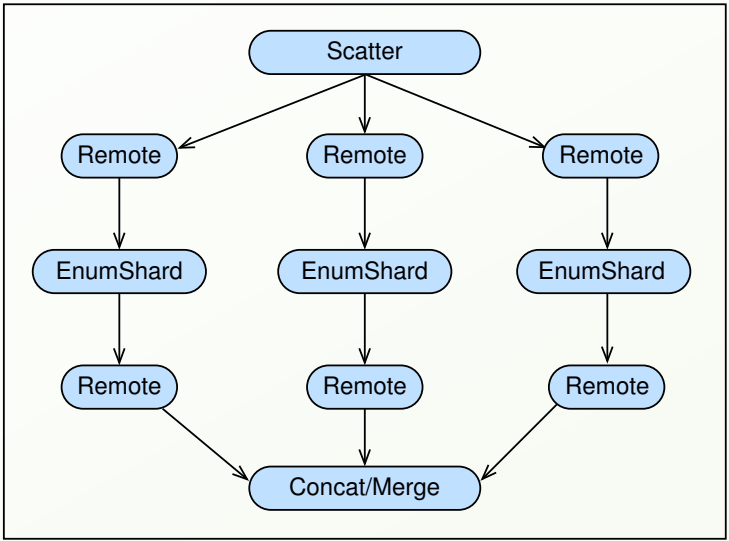


- The shards of a collection are distributed across the DB servers.
- The coordinators receive queries and organise their execution

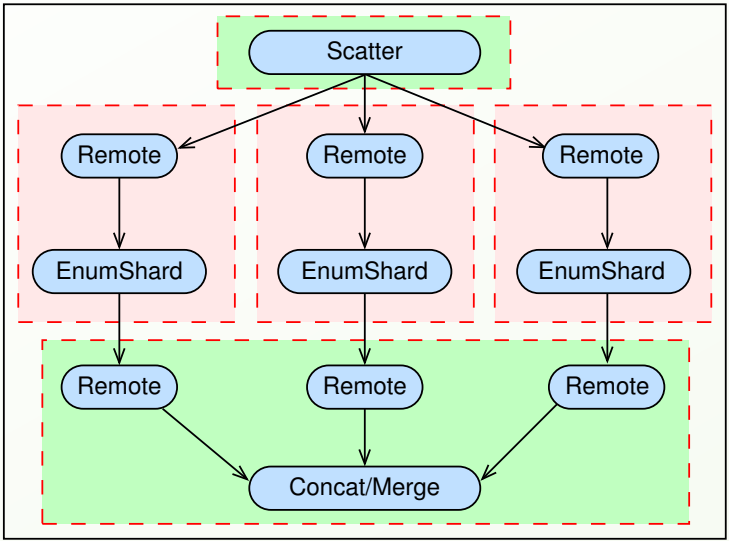
Scatter/gather



Scatter/gather



Scatter/gather



Links

https://www.arangodb.com

https://docs.arangodb.com/cookbook/index.html

https://github.com/ArangoDB/guesser

http://mesos.apache.org/

https://mesosphere.com/

https://mesosphere.github.io/marathon/

https://dcos.io