Putting the C back in CouchDB

Joan Touzet - wohali

© TemplatesWise.com

WHO AM I?

CouchDB

- Contributor / User (~2008)
- Committer (Feb 2013)
- PMC member (April 2014)

IBM Cloudant

- Engineer (2012-2013)
- Sr. SW Development Manager (2014-)

SO MUCH TO DISCUSS…

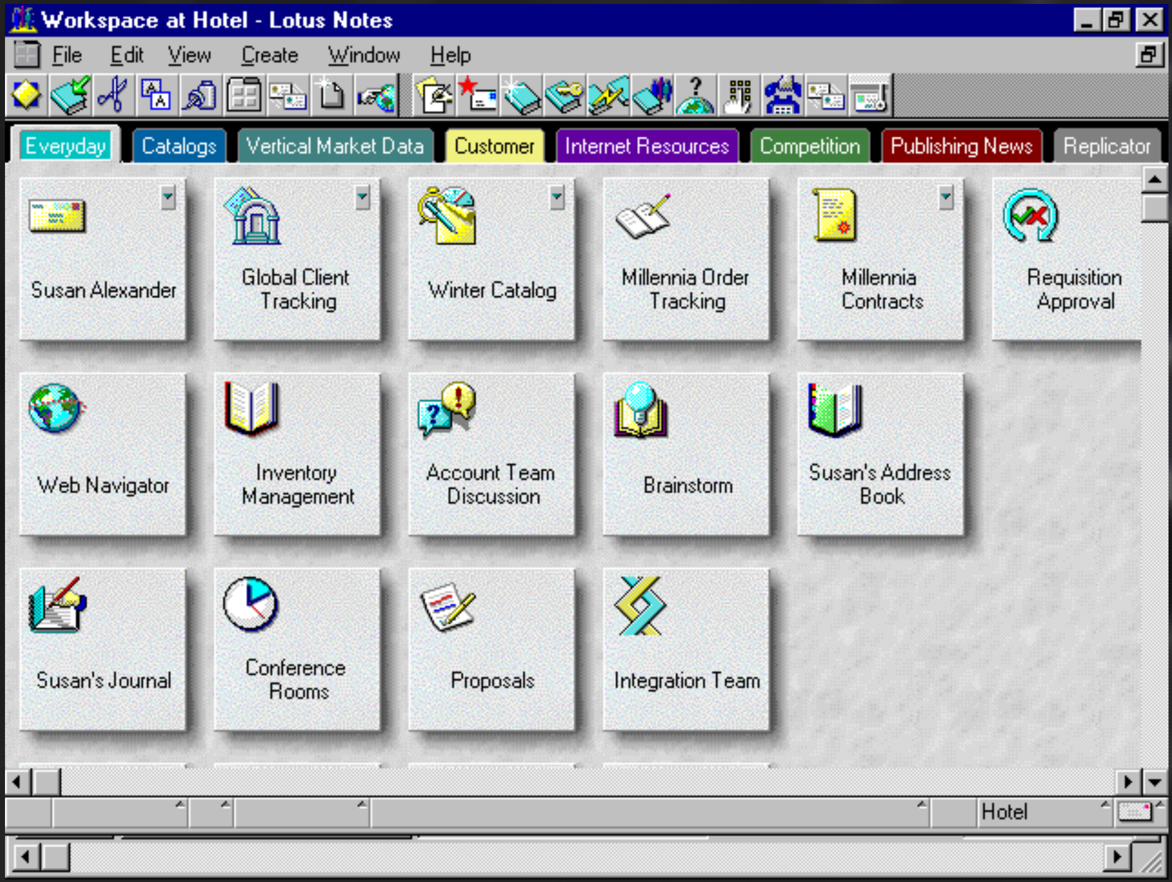
This talk is focused on clustering in CouchDB 2.0.

I'm sneaking in slides on Query as well.

I'm happy to discuss any other new 2.0 features during the Q&A portion of the talk.

PART 1: MOTIVATION

1989: “Non-SQL bi-directional synchronization”

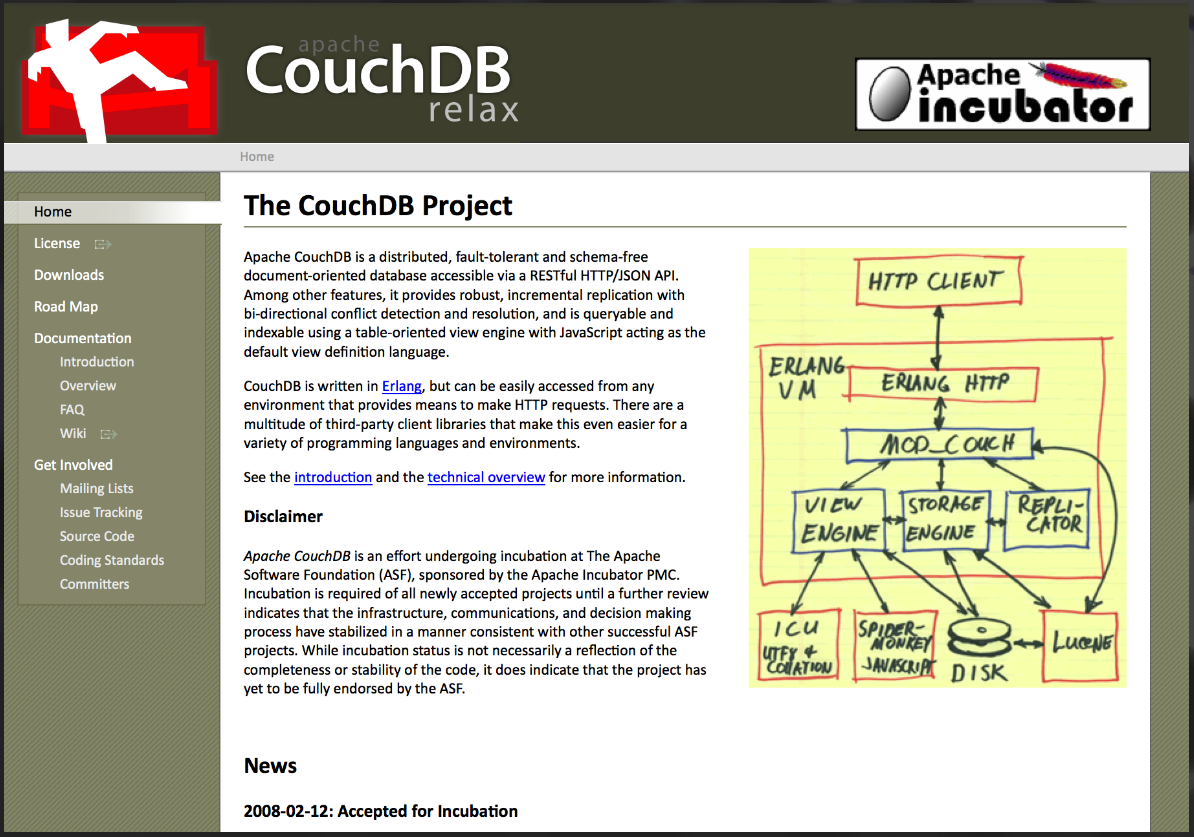


2005: bi-directional synchronization reborn

Apache Incubator in Feb 2008

Top Level Project in Nov 2008

1.0 release in July 2010



apache CouchDB relax

Apache incubator

Home

- Home
- License
- Downloads
- Road Map
- Documentation
  - Introduction
  - Overview
  - FAQ
  - Wiki
- Get Involved
  - Mailing Lists
  - Issue Tracking
  - Source Code
  - Coding Standards
  - Committers

The CouchDB Project

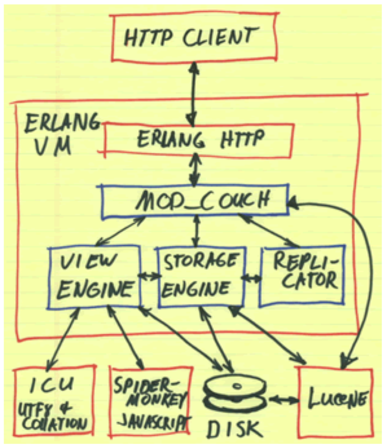
Apache CouchDB is a distributed, fault-tolerant and schema-free document-oriented database accessible via a RESTful HTTP/JSON API. Among other features, it provides robust, incremental replication with bi-directional conflict detection and resolution, and is queryable and indexable using a table-oriented view engine with JavaScript acting as the default view definition language.

CouchDB is written in Erlang, but can be easily accessed from any environment that provides means to make HTTP requests. There are a multitude of third-party client libraries that make this even easier for a variety of programming languages and environments.

See the introduction and the technical overview for more information.

Disclaimer

Apache CouchDB is an effort undergoing incubation at The Apache Software Foundation (ASF), sponsored by the Apache Incubator PMC. Incubation is required of all newly accepted projects until a further review indicates that the infrastructure, communications, and decision making process have stabilized in a manner consistent with other successful ASF projects. While incubation status is not necessarily a reflection of the completeness or stability of the code, it does indicate that the project has yet to be fully endorsed by the ASF.



News

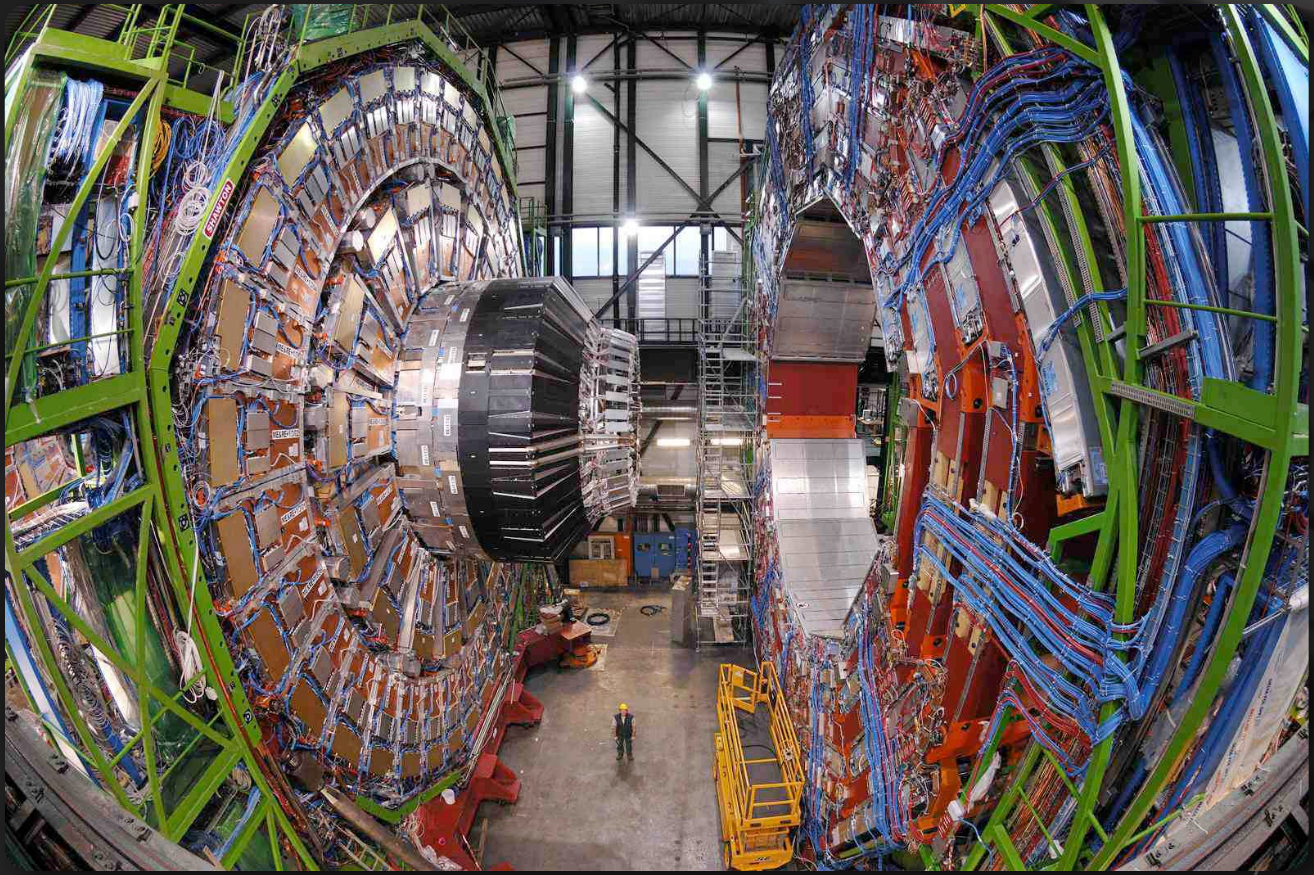
2008-02-12: Accepted for Incubation

2010: CMS Detector, LHC, CERN

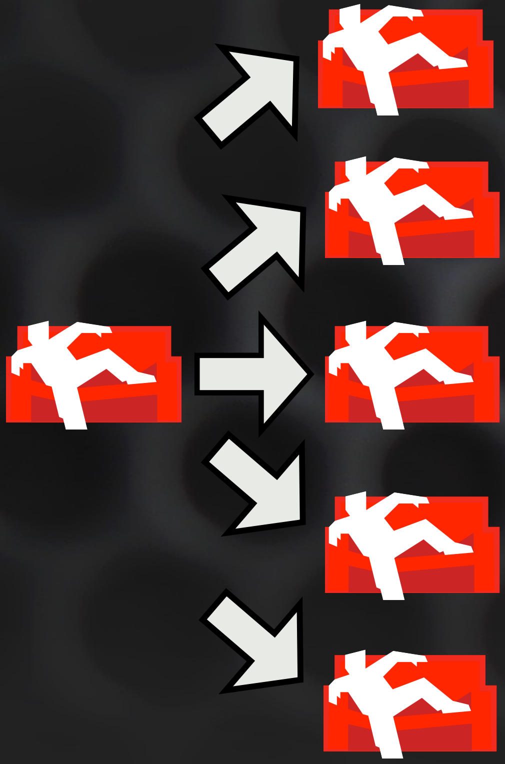
In 2010, adopted CouchDB

Est. 10 petabytes / year

Built & operated by:
- 3800 people
- from 182 institutes
- in 42 countries
- …who all need the data!



But it wasn't enough…



But it wasn't enough...

Cluster

Of

Unreliable

Commodity

Hardware

Sunburned by Emily Hildebrand

EVOLVE OR PERISH!

Source: Sony Online Entertainment (Used with permission)

PART 2: CLUSTERING

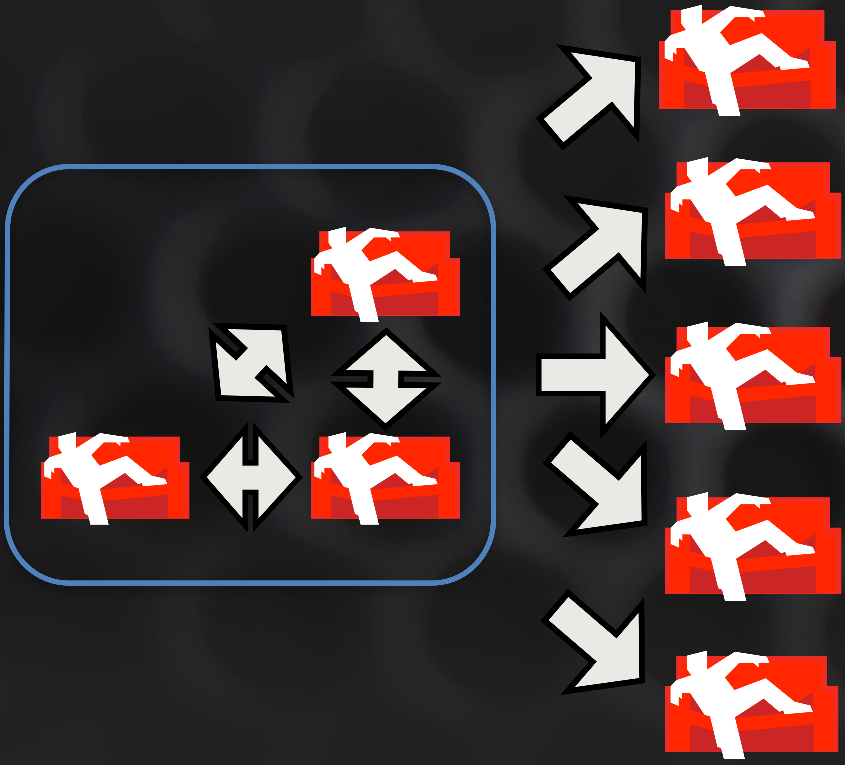
CouchDB needed scaling.

Vertical scaling (bigger single server) has upper bounds and is a Single Point of Failure (SPOF).

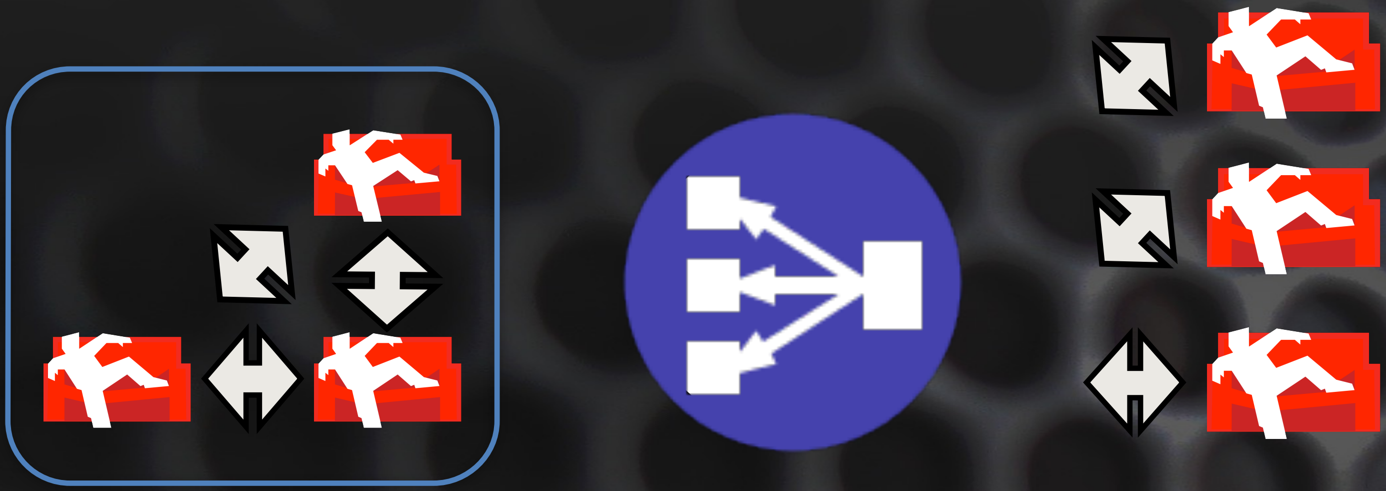
Horizontal scaling (more servers in parallel) creates more true capacity.

Transparent to the application: adding more capacity should not affect the business logic of the application.

What if…



The BigCouch Solution

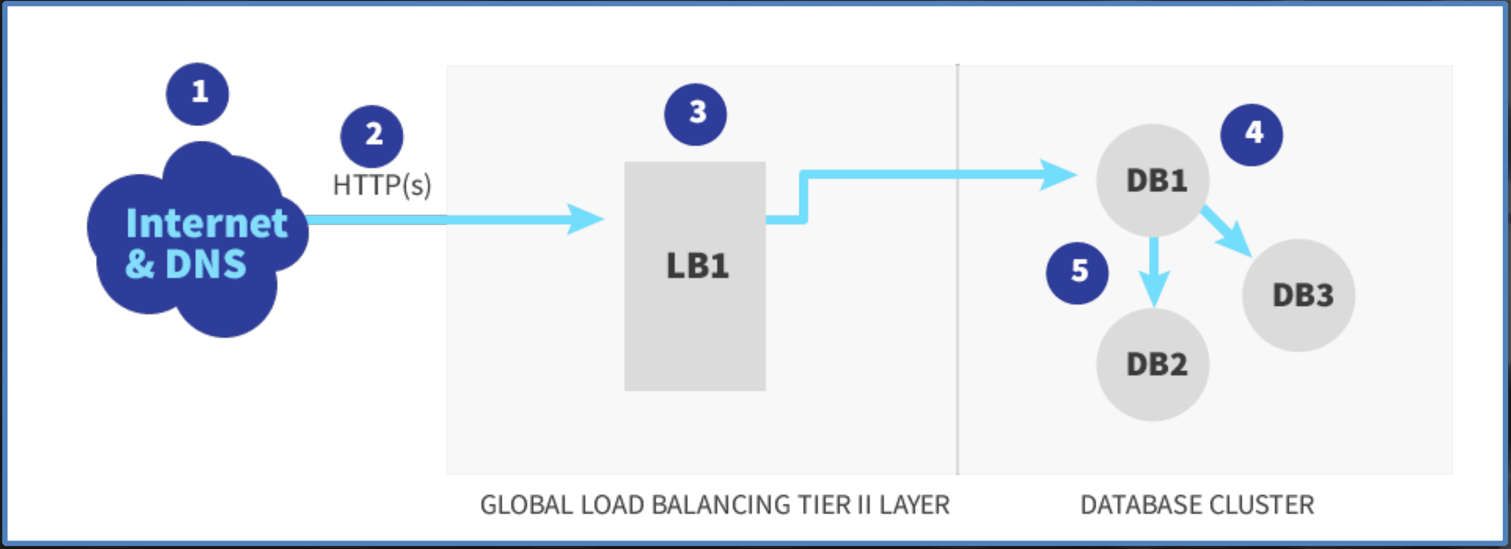


BigCouch Cluster

Load Balancer (haproxy)

Clients (same as always!)

The ~~BigCouch~~ Clustered Solution



The Clustered Solution

Easily add more storage with more cluster nodes

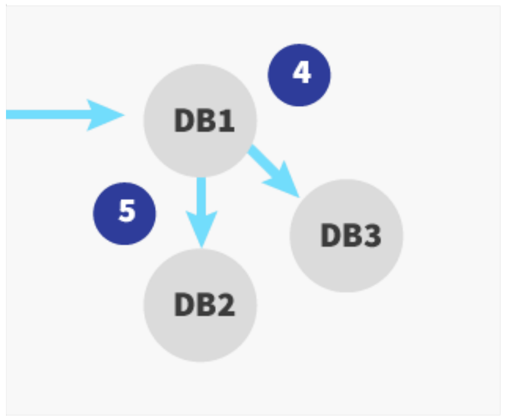
Compute power (indexes, compaction, etc.) scales linearly with number of nodes

No SPOFs: nodes can come and go

Clustering is entirely transparent to the application

Can optimize intra-cluster communication

(*Caveats will be discussed.*)



DATABASE CLUSTER

Clustering Parameters

Terminology comes from the 2007 Amazon Dynamo paper:
http://www.allthingsdistributed.com/2007/10/amazons_dynamo.html

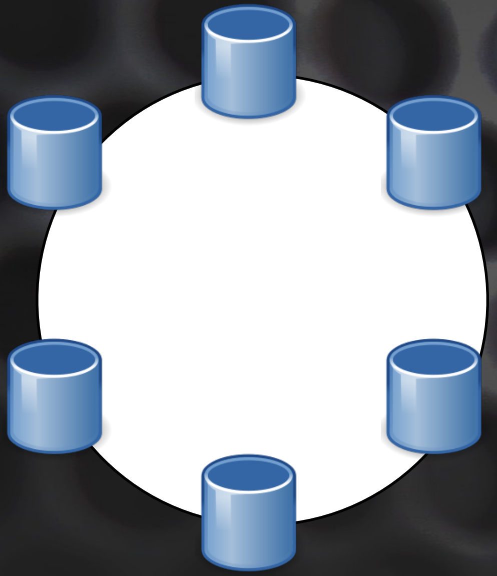
Nodes - # of machines in the cluster

N - # of copies/replicas of the data

Q - # of unique shards for a database

R - read quorum

W - write quorum

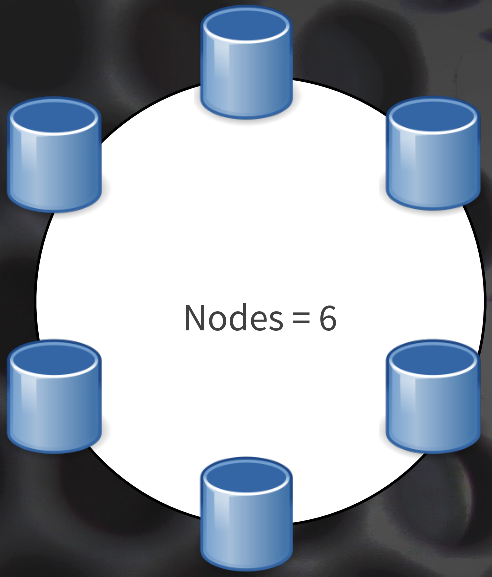


# of Nodes

Typically multiples of 3

Nodes = 3, Nodes = 6, Nodes = 18, Nodes = 24…

Nodes = 1 still supported!



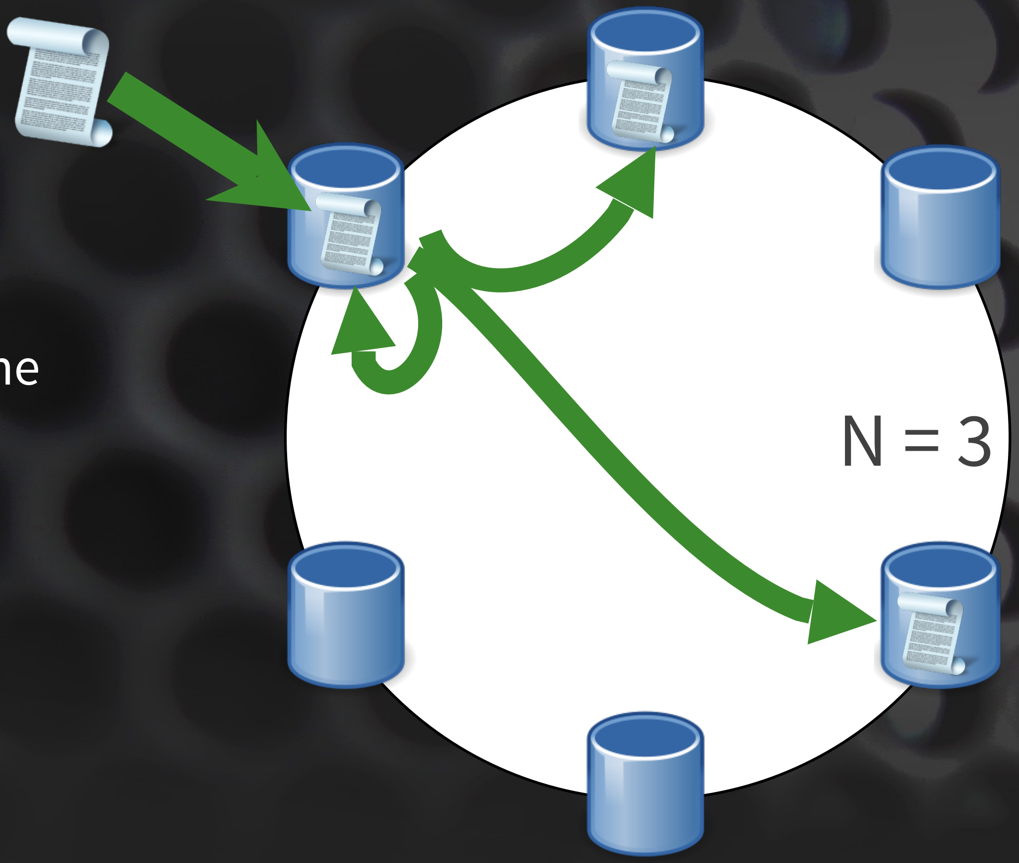
N - # of copies/replicas of the data

On write, store **N** copies of data

Configurable per DB at creation time

Default is 3

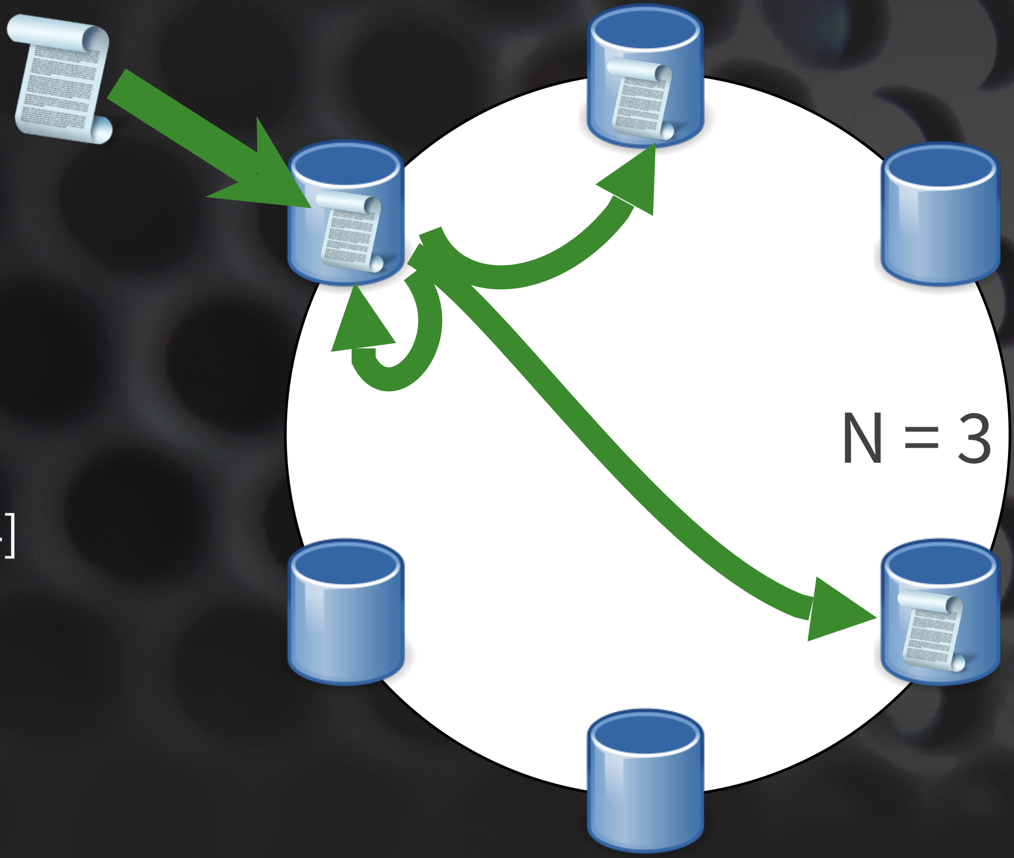
Rarely changed



N - # of copies/replicas of the data

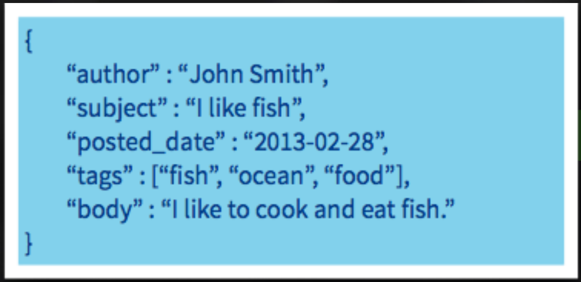
Node Computes:

1. key = hash(doc._id)
2. get_shards(key) ==> shard
3. get_nodes(shard) ==> [N1,N3,N4]
4. Nodes.foreach: store(doc)

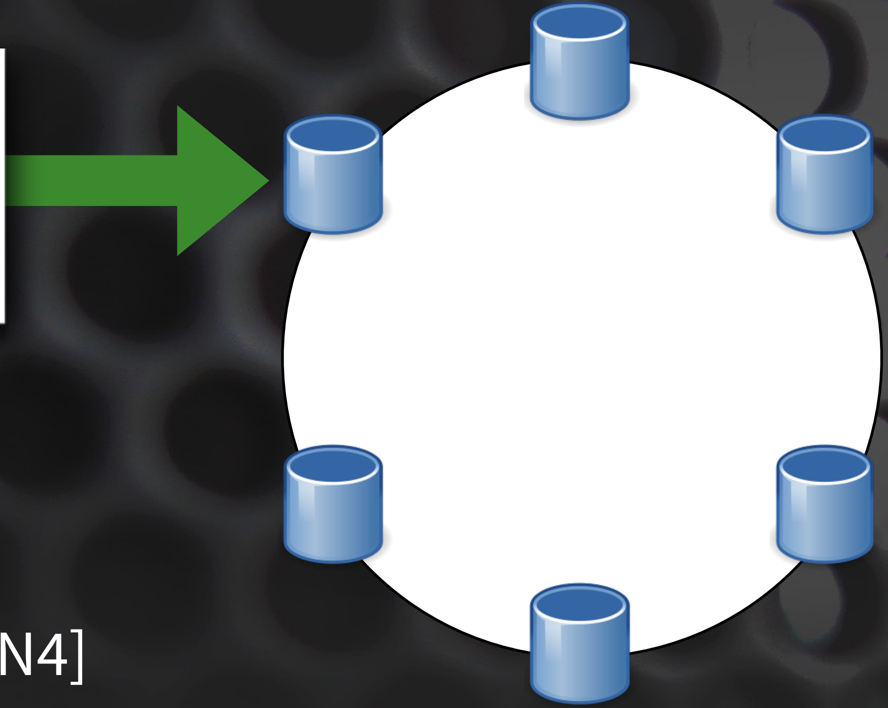


What are shards?

```
PUT /db7/docid92
```



```
{
    "author" : "John Smith",
    "subject" : "I like fish",
    "posted_date" : "2013-02-28",
    "tags" : ["fish", "ocean", "food"],
    "body" : "I like to cook and eat fish."
}
```



Which nodes get docid92?

1. key = hash("docid92")
2. get_shards(key) ==> shard
3. get_nodes(shard) ==> [N1,N3,N4]
4. Nodes.foreach: store(doc)

Q - # of unique shards for a database

Default is 8

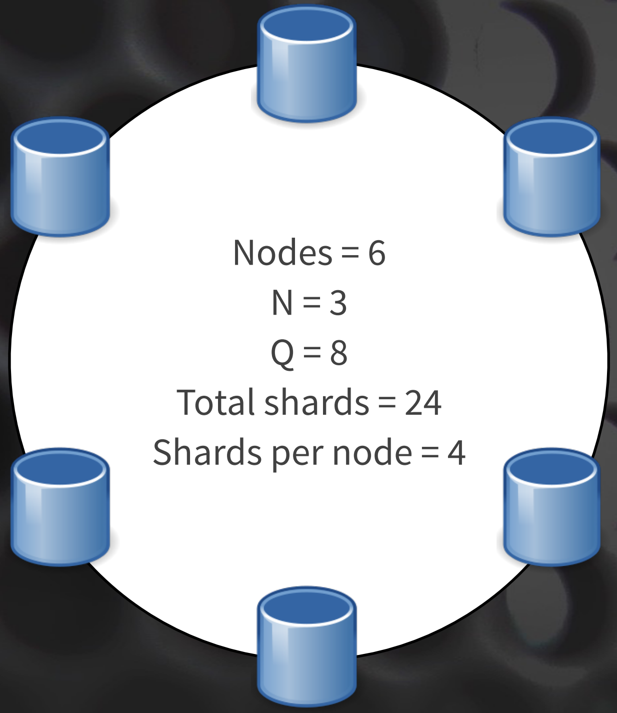
Configurable per DB at creation time

Total Shards = Q × N

   Default = 8 × 3 = **24**

Recommend Q is a multiple of # of nodes

Q sets your degree of parallelism



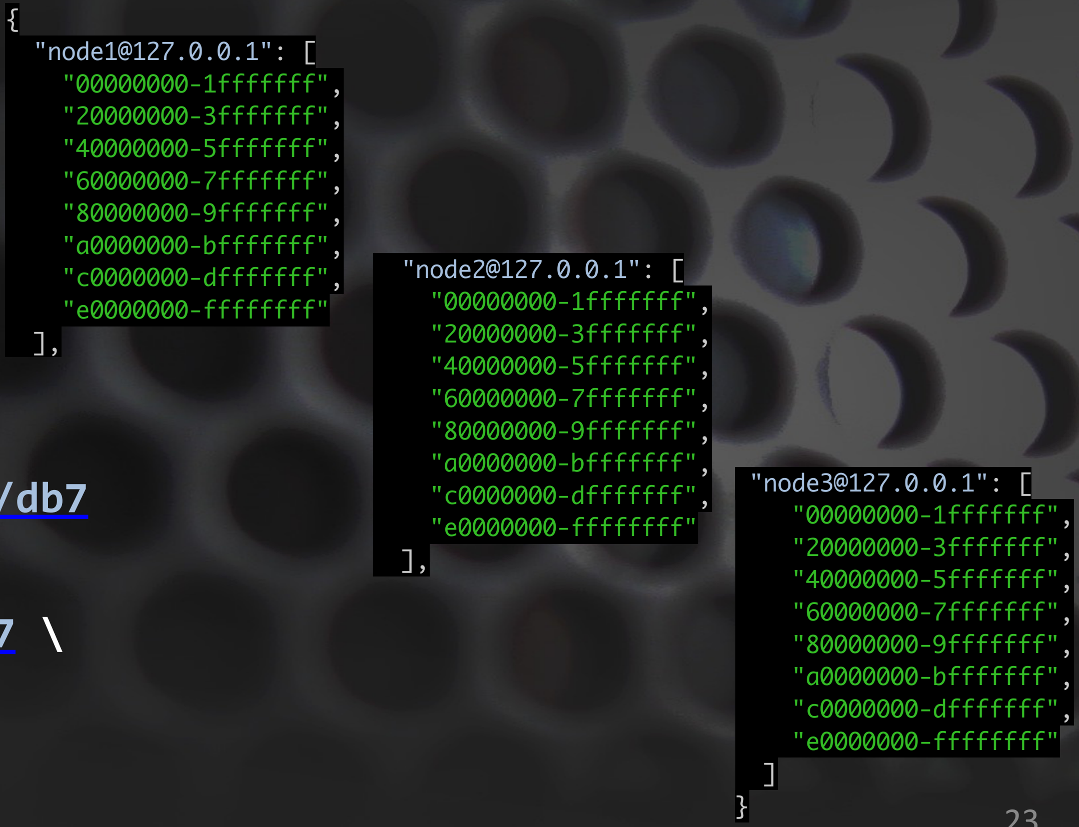
Example Shard Map

Q = 8, N = 3, Nodes = 3

means 24 shards, 8 on each node

See for yourself on a dev setup:

```
# install jq, then:
$ curl -X PUT http://localhost:15984/db7
{"ok": true}
$ curl http://localhost:15986/dbs/db7 \
  | jq .by_node
```



```
{
  "node1@127.0.0.1": [
    "00000000-1fffffff",
    "20000000-3fffffff",
    "40000000-5fffffff",
    "60000000-7fffffff",
    "80000000-9fffffff",
    "a0000000-bfffffff",
    "c0000000-dfffffff",
    "e0000000-ffffffff"
  ],
  "node2@127.0.0.1": [
    "00000000-1fffffff",
    "20000000-3fffffff",
    "40000000-5fffffff",
    "60000000-7fffffff",
    "80000000-9fffffff",
    "a0000000-bfffffff",
    "c0000000-dfffffff",
    "e0000000-ffffffff"
  ],
  "node3@127.0.0.1": [
    "00000000-1fffffff",
    "20000000-3fffffff",
    "40000000-5fffffff",
    "60000000-7fffffff",
    "80000000-9fffffff",
    "a0000000-bfffffff",
    "c0000000-dfffffff",
    "e0000000-ffffffff"
  ]
}
```

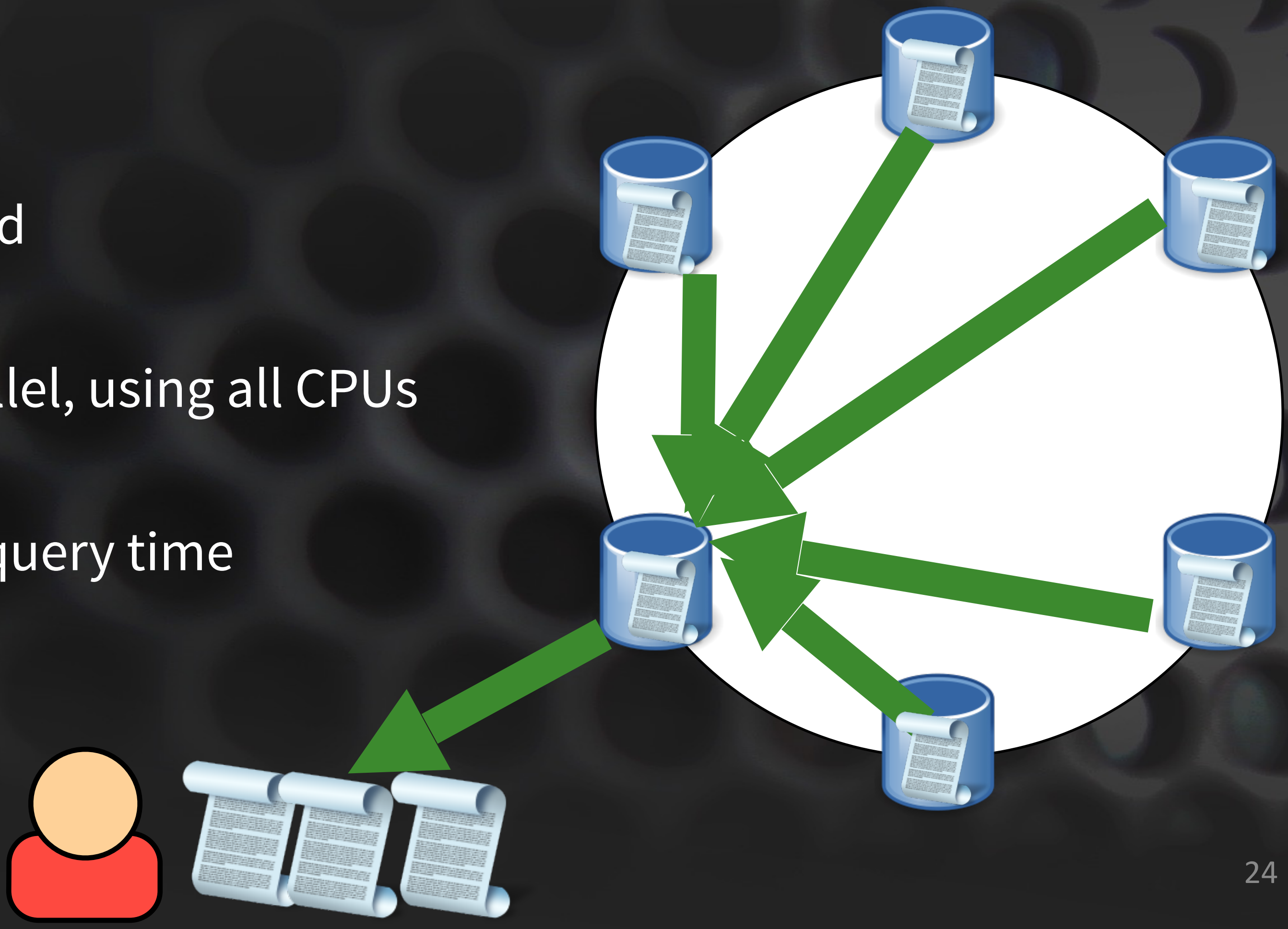
Try adding ?q=4 to the PUT, or add 3 more nodes!

How do indexes work?

Built locally for each shard

View shards build in parallel, using all CPUs

Merge-sort responses at query time



“But how do I pick Q?”

General Rule:

- **If the cluster has just a few large DBs, use large Q.**
- **If the cluster has many small DBs, use small Q.**

# of shards defines your degree of parallelism.

- Consider the number of disk spindles & CPU cores in the cluster.

Each shard file should be 10GB or less.

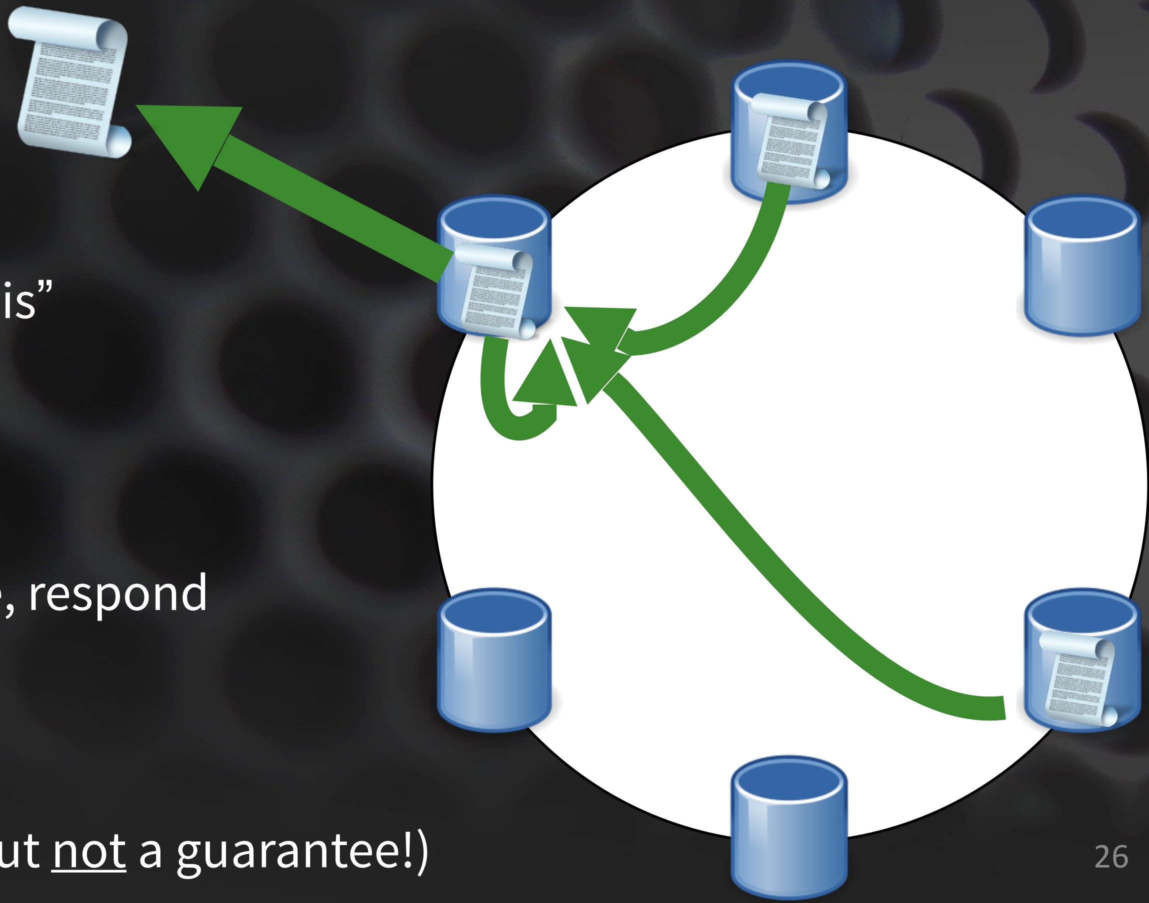
- Bigger shard files can adversely affect compaction.

Large # of writes at load will require more shards.

When all else fails, experiment with different values under load.

R - Read Quorum

```
GET /db7/docid92
```



When does DB say “here it is”?

⇛ When enough nodes say “here it is”

What is “enough”?

⇛ Try to read it from N Nodes

⇛ When “R” nodes reply and agree, respond

Default: R = 2 (majority)

R = 1 will minimise latency

R = N will maximise consistency (but <u>not</u> a guarantee!)

W - Write Quorum

`PUT /db7/docid92`

When does DB say “written”?

⇛ When enough nodes have written

What is “enough”?

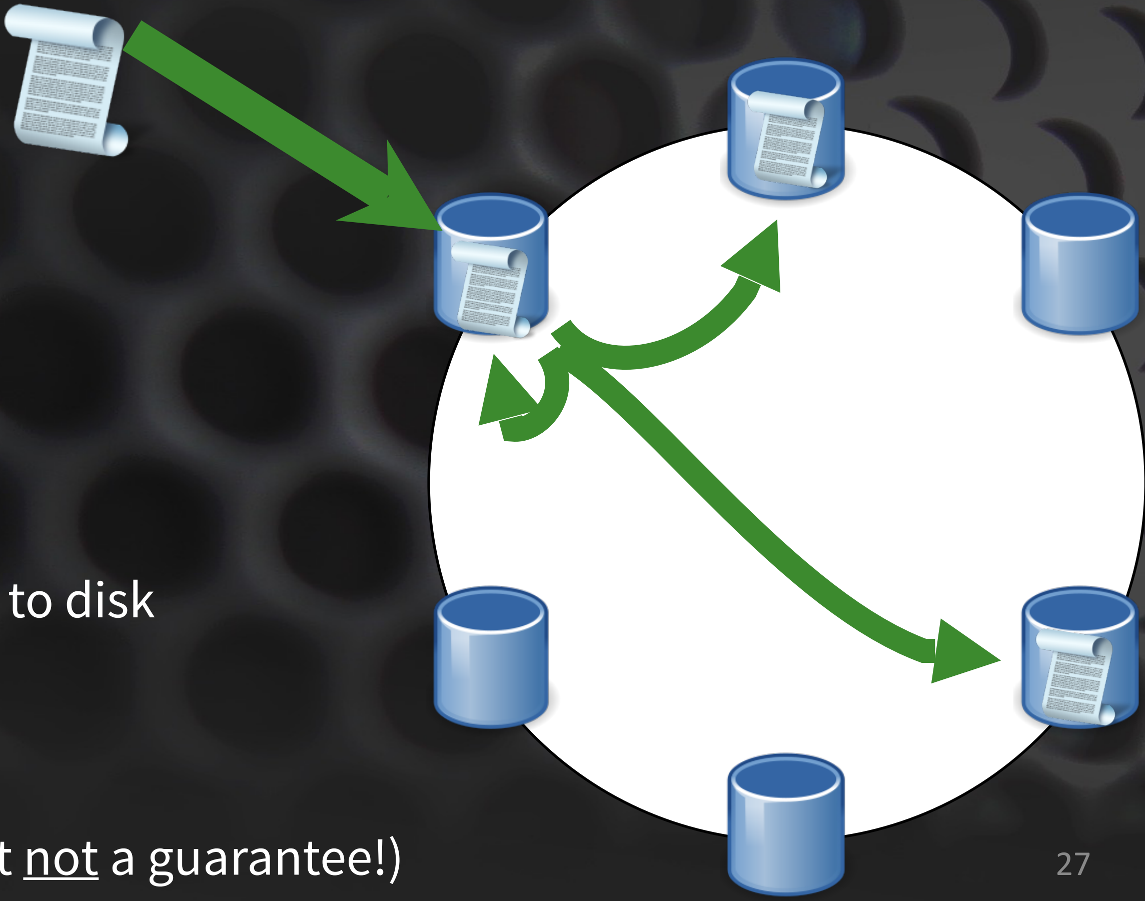
⇛ Try to store all replicas (N copies)

⇛ When “W” nodes reply, after fsync to disk

Default: W = 2 (majority)

W = 1 will maximise latency

W = N will maximize consistency (but not a guarantee!)



Read and Write Quorum

r can be specified at query time, w can be specified at write time

Inconsistencies are repaired at read time

Pay attention to your HTTP status codes & returned messages!

- 200 – OK
- 201 – wrote successfully, quorum met
- 202 – quorum on write wasn’t met || batch mode || bulk with conflicts
- 400 – format was invalid
- 403 – unauthorized
- 404 – resource not found
- 409 – document conflict, or no rev specified
- 412 – database already exists

Caveats

`_changes` feed works similarly to CouchDB 1.x, but has no global ordering

- CouchDB is an AP system, not a CP system!

Clustered API listens on port 5984

- `by_sequence` key is now an opaque string, not an integer.
- `rereduce=true` for all MapReduce views, always

'Backdoor' access listens on port 5986

- Able to reach a single node (i.e. at the shard level)
- Allows you to trigger local view updates, compactions, etc.

Part 3: Query

Introducing Query

New declarative query language for accessing your data

Easy for developers to learn and use when coming from a SQL world

Establishing a NoSQL Document Database standard based on MongoDB’s query language syntax

```
{
  "index": {
    "fields": ["foo"]
  },
  "name": "foo-index",
  "type": "json"
}
```

```
{
  "selector": {
    "bar": {"$gt": 1000000}
  },
  "fields": ["_id", "_rev", "foo", "bar"],
  "sort": [{"bar": "asc"}],
  "limit": 10,
  "skip": 0
}
```

EVOLVE OR PERISH!

Source: Sony Online Entertainment (Used with permission)

Query Technical Overview

Two new API endpoints: **`/_index`** and **`/_find`**

Query indexes are implemented as MapReduce functions behind the scenes

- Natively compiled in Erlang versus interpreted JavaScript functions

It is NOT a 1:1, fully-compatible mapping with MongoDB

- Fields must be indexed before they can be queried
- Extra functionality, such as aggregation, is not available but is a likely addition to future versions

Full docs available today at https://docs.cloudant.com/api/cloudant-query.html

Query Comparison 4 Ways

SQL

```
SELECT *
FROM people
WHERE age > 25
AND age <= 50;
```

MongoDB

```
db.people.find(
  { age: { $gt: 25, $lte: 50 } }
)
```

CouchDB MapReduce view

1) Create design document:

```
{
  "_id": "_design/userview",
  "views": {
    "byAge": {
      "map": "function(doc){\n\t if (doc.type==\"user\" && doc.age) {\n\t\t emit(doc.age, null);\n\t}\n}" }
  },
  "language": "javascript"
}
```

2) Wait for view to build
3) Command line:

```
curl http://localhost:5984/people/_design/userview/_view/byAge?startkey=25&endkey=50
```

Query Comparison 4 Ways

Query

```
curl -X POST 'http://localhost:5984/users/_find' -d
'{
  "selector": {
    "age": {
      "$gt": 25,
      "$lte": 50
    }
  }
}'
```

Creating a new Query Index

POST http://localhost:5984/<database>/**_index**

Create an index in a specified DB by **POST**ing an appropriate JSON object to the /<database>/**_index** endpoint

All fields included in the indexing request then become searchable through the **_find** URL endpoint

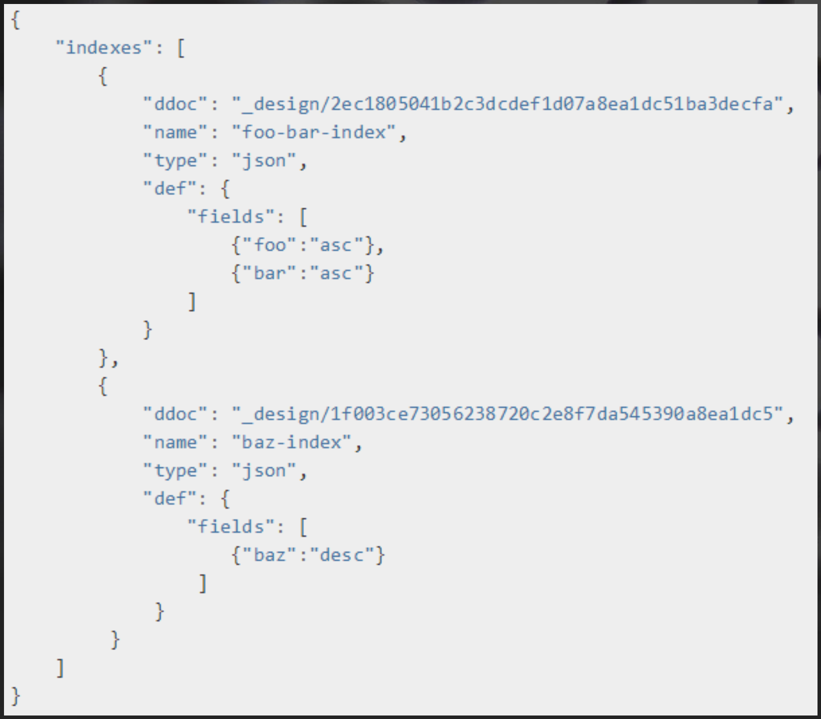
```
POST /db/_index
Content-Type: application/json

{
  "index": {
    "fields": ["foo"]
  },
  "name": "foo-index",
  "type": "json"
}
```

Retrieving Index Information

GET http://localhost:5984/<database>/**_index**

Returns a list of all indexes in a specified DB with a **GET** request to a specific /<database>/**_index** endpoint

Each index created using Query is placed in its own design document with a unique identifier



```
{
    "indexes": [
        {
            "ddoc": "_design/2ec1805041b2c3dcdef1d07a8ea1dc51ba3decfa",
            "name": "foo-bar-index",
            "type": "json",
            "def": {
                "fields": [
                    {"foo":"asc"},
                    {"bar":"asc"}
                ]
            }
        },
        {
            "ddoc": "_design/1f003ce73056238720c2e8f7da545390a8ea1dc5",
            "name": "baz-index",
            "type": "json",
            "def": {
                "fields": [
                    {"baz":"desc"}
                 ]
             }
         }
    ]
}
```

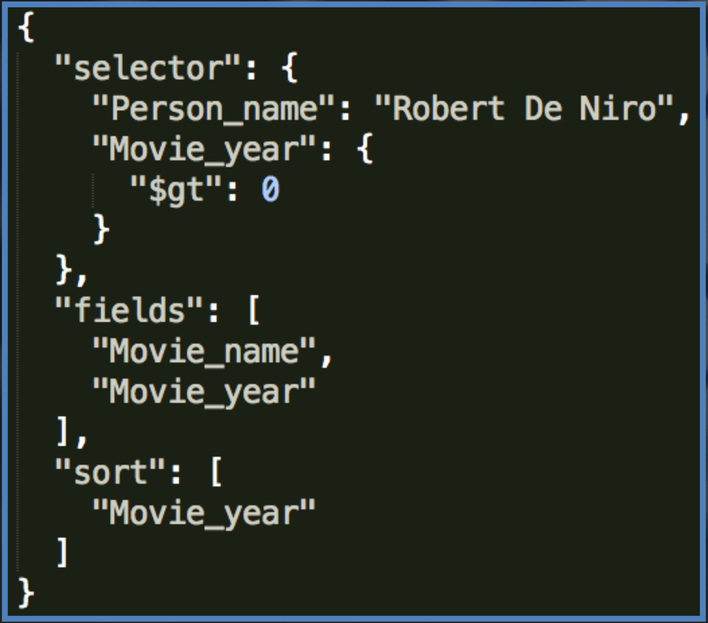
Executing a Query

POST http://localhost:5984/<database>/**_find**

Query against a database's index by **POST**ing to the /<database>/**_find** endpoint

The JSON must contain a selector object, and can contain any of these optional parameters:

```
  fields, sort, limit, skip, r
```



```
{
  "selector": {
    "Person_name": "Robert De Niro",
    "Movie_year": {
      "$gt": 0
    }
  },
  "fields": [
    "Movie_name",
    "Movie_year"
  ],
  "sort": [
    "Movie_year"
  ]
}
```

Sorting and Filtering a Query

Sort with a basic array of fields and direction parameters.
One sort field must be in the selector. All sort fields must be indexed.

```
curl -X POST 'https://<accountname>.cloudant.com/movies/_find' -d
'{
  "selector": {
    "Actor_name": "Robert De Niro",
    "Movie_year": {"$gt": 1960}
  },
  "sort": [{"Actor_name": "asc"}, {"Movie_runtime": "asc"}]
}'
```

Filtering returns a subset of fields. _id and _rev are not automatic.
Filtering fields do not have to be indexed.

```
curl -X POST 'https://<accountname>.cloudant.com/movies/_find' -d
'{
  "fields": ["Movie_name", "Movie_year"],
  "selector": {
    "Person_name": "Alec Guinness",
    "Movie_year": {"$gt": 1960}
  }
}'
```

Refining a Query

Query against an index and refine the result set by applying conditions on fields *beyond* the original index.

Find all De Niro films from a specific year (1978)

```
{
  "selector": {
    "Person_name": "Robert De Niro",
    "Movie_year": 1978
  }
}
```

```
{
  "docs": [
    {
      "Movie_genre": "DW",
      "Movie_name": "Deer Hunter, The",
      "Movie_rating": "R",
      "Movie_runtime": 183,
      "Movie_year": 1978,
      "Person_dob": "1943-08-17",
      "Person_name": "Robert De Niro",
      "Person_pob": "New York, New York, USA",
      "_id": "1f003ce73056238720c2e8f7da428f32",
      "_rev": "1-3fa59b11f43719f46c288b9bb9943d1d"
    }
  ]
}
```

In this example, only `Person_name` is indexed. If you select on a field often, **index it**.

Some notes…

You decide which fields are indexed. They are not created automatically.

Selector syntax supports combination and condition operators.

```
{ “name”: “Paul” }  ⟺  { “name”: { “$eq”: “Paul” } }

{ “name”: “Paul”, “location”: “Boston” }

{ “location”: { “city”: “Omaha” } }  ⟺  { “location.city”: “Omaha” }

{ “age”: { “$gt”: 20 } }
```

Query Combination Operators

‘Combination Operators’ take a single argument (either a selector or an array of selectors) for combination

Operator	Usage
$and	Matches if all selectors in the array match
$or	Matches if any selectors in the array match
$not	Matches if the given selector does not match
$nor	Matches if none of the selectors (multiple) match
$all	Matches an array value if it contains all element of argument array
$elemMatch	Returns first element (if any) matching value of argument

‘Condition Operators’ (next slide) are specified on a per-field basis, and apply to the value indexed for that field.

Query Condition Operators

Operator	Usage
$lt	Less than
$lte	Less than or equal to
$eq	Equal to
$ne	Not equal to
$gt	Greater than
$gte	Greater than or equal to
$exists	Boolean (exists or it does not)
$type	Check document field’s type
$in	Field must exist in the provided array of values
$nin	Field must not exist in the provided array of values
$size	Length of array field must match this value
$mod	[Divisor, Remainder]. Returns true when the field equals the remainder after being divided by the divisor.
$regex	Matches provided regular expression

Thank you for listening!

couchdb.apache.org

github.com/apache/couchdb

@wohali