Benchmarking MongoDB and Couchbase

No-SQL Databases

Alex Voss **Chris Choi**

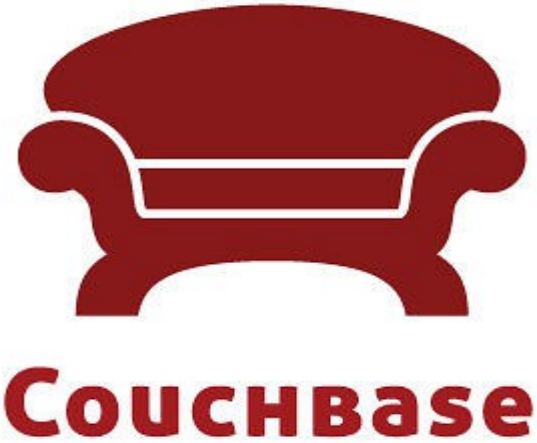
University of St Andrews

TOP 2 Questions

Should a social scientist *buy*
MORE or UPGRADE
computers?

Which DATABASE(s)?

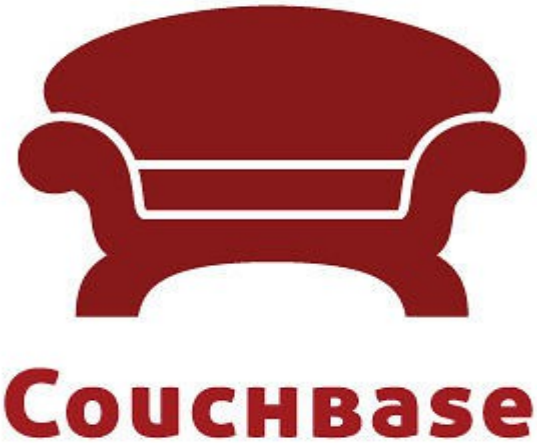
Document Oriented Databases



The central concept of a document-oriented database is the notion of a Document

Both

and



Uses JSON to store these Documents

Example Documents:

```
{
    FirstName:"Bob",
    Address:"5 Oak St.",
    Hobby:"sailing"
}
```

```
{
    FirstName:"Jonathan",
    Address:"15 Wanamassa Point Road",
    Children:[
        {Name:"Michael",Age:10},
        {Name:"Jennifer", Age:8},
        {Name:"Samantha", Age:5},
        {Name:"Elena", Age:2}
    ]
}
```

there are no empty 'fields', and it does not require explicitly stating if other pieces of information are left out

The Database Tests

Database Test

Aggregate/Count:

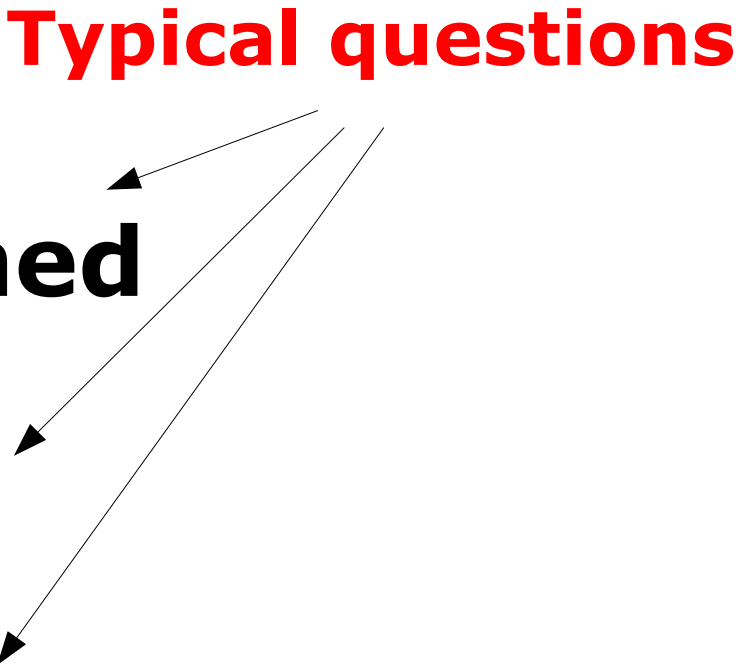
Most User Mentioned

Most Hashed Tags

Most Shared URLs

Database Test

Aggregate/Count:



Most User Mentioned

Most Hashed Tags

Most Shared URLs


mongoDB

Has two query approaches:

Map Reduce **&** **Aggregation Framework**

Map Reduce

For ***Aggregation***

Uses **JavaScript**

Aggregation Framework

Similar to Map Reduce

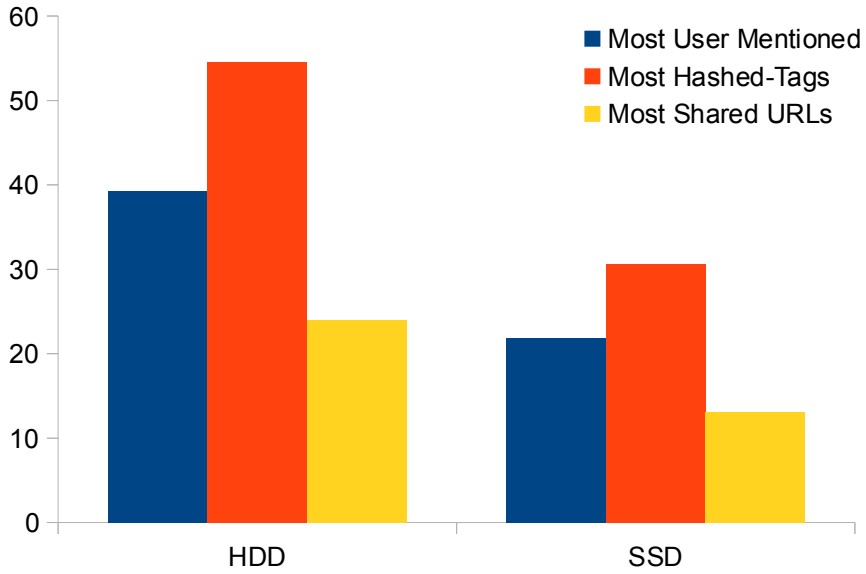
But Uses C++ (Faster!)

mongoDB

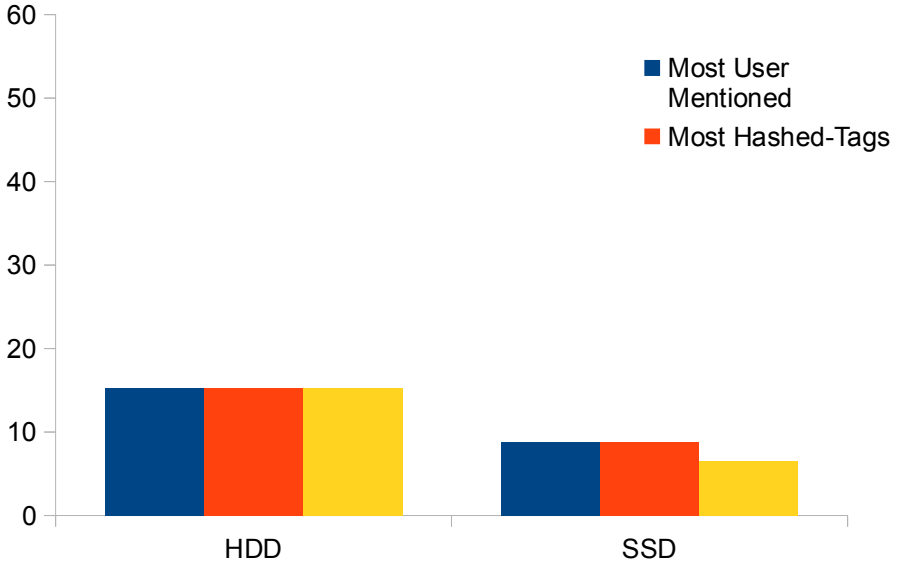
Single Node

SSD vs HDD

Map Reduce



Aggregation Framework



Single Node

Aggregation Framework

is roughly *2 times faster* than

Map Reduce

When we have more than 1 node ~ Scaling

Generally . . .

Scaling Vertically

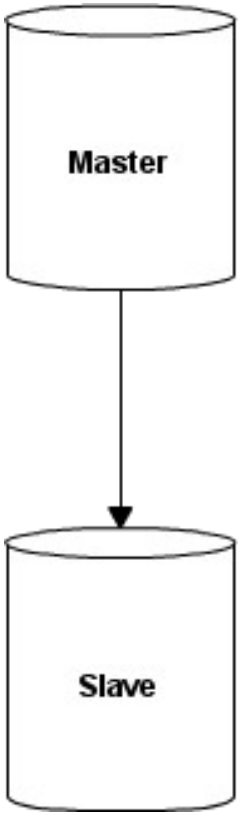
BIGGER and **FATTER** machine!

Scaling Horizontally

MORE machines!

Replica-Set

Master-Slave Replication

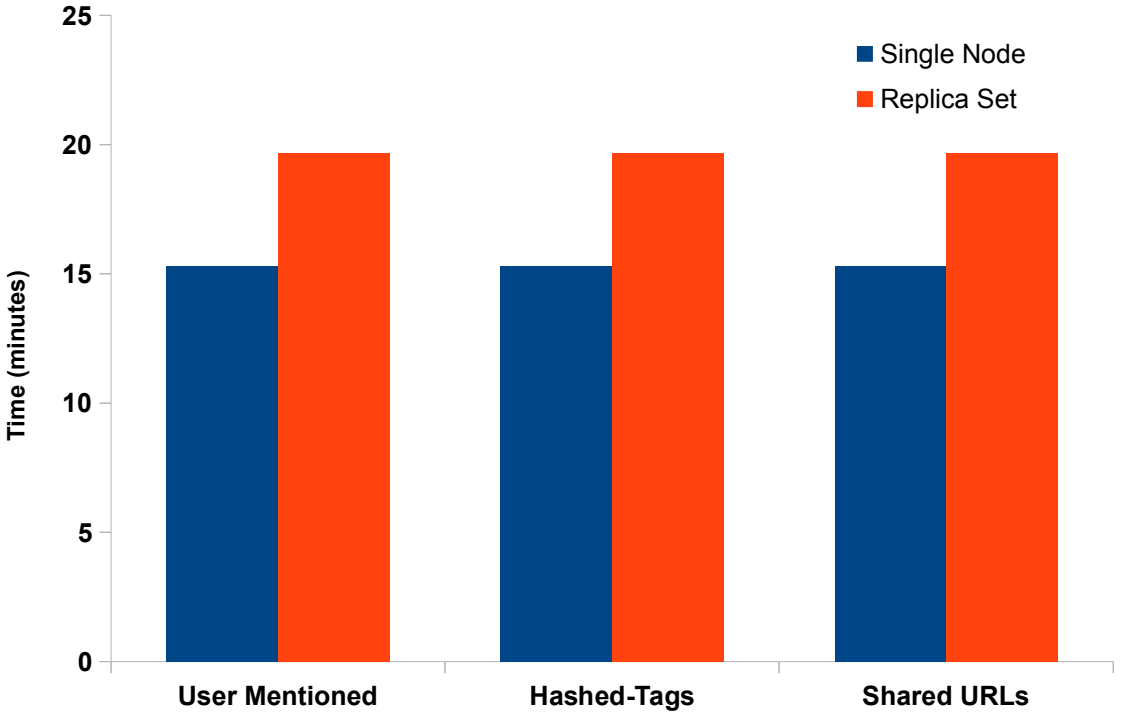


Replicates data from master to slave

Only one node is in charge (the Master), and data only travels in one direction (to the Slave)

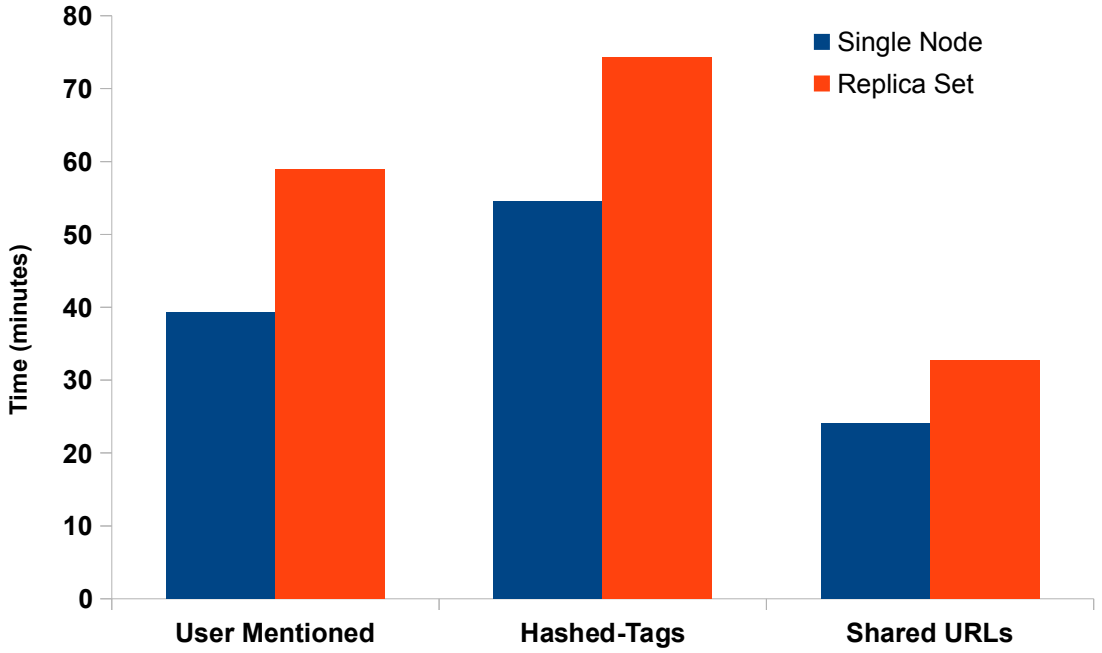
Replica-Set

Single Node vs Replica Set: Aggregation Framework



Replica-Set

Single Node vs Replica Set: Map Reduce



***Replicating Data* did not improve query performance**

(MapReduce and Aggregation Framework) . . .

so we tried

Sharding

Sharding

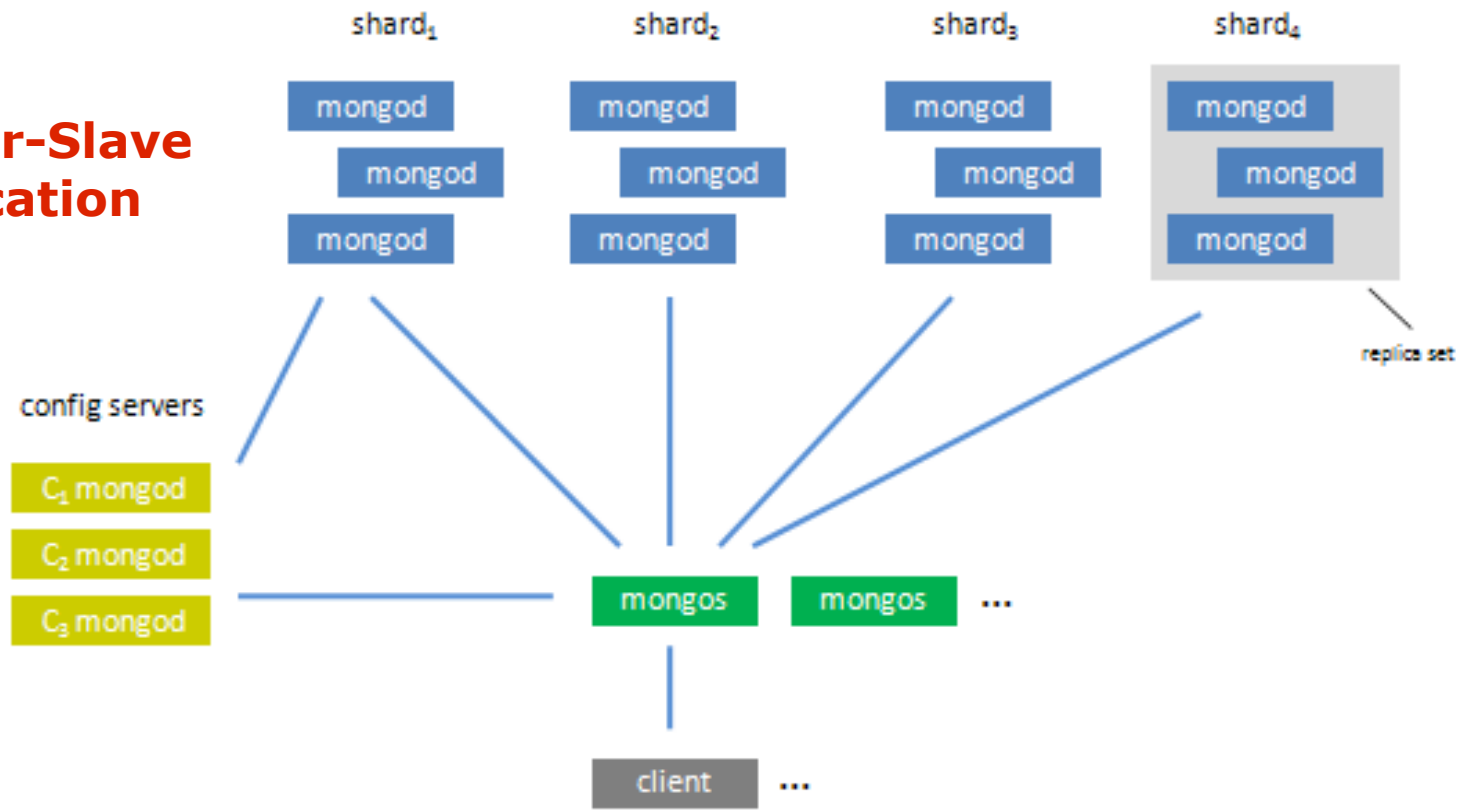
One BIG stack

Two or more smaller stacks

mongoDB

uses a democratic hierarchical scaling approach

Master-Slave Replication



But Deployment is an issue with MongoDB . . .

If you want to shard . . .

documentation **suggests**

minimum **11** **nodes, to build a**

fail-safe ***sharded cluster***

2 Shards → 6 nodes (3 each)

2 `mongos` (load balancer) → 2 nodes

3 `mongod` (config servers) → 3 nodes

Total: 11 nodes

The *investment* to go from single node --> sharded cluster is very HIGH!

By omitting Fail-Safe features

For our tests, we kind of *cheated* and used:

Note: X is variable

~~2~~ X Shards → ~~6~~ 2 nodes

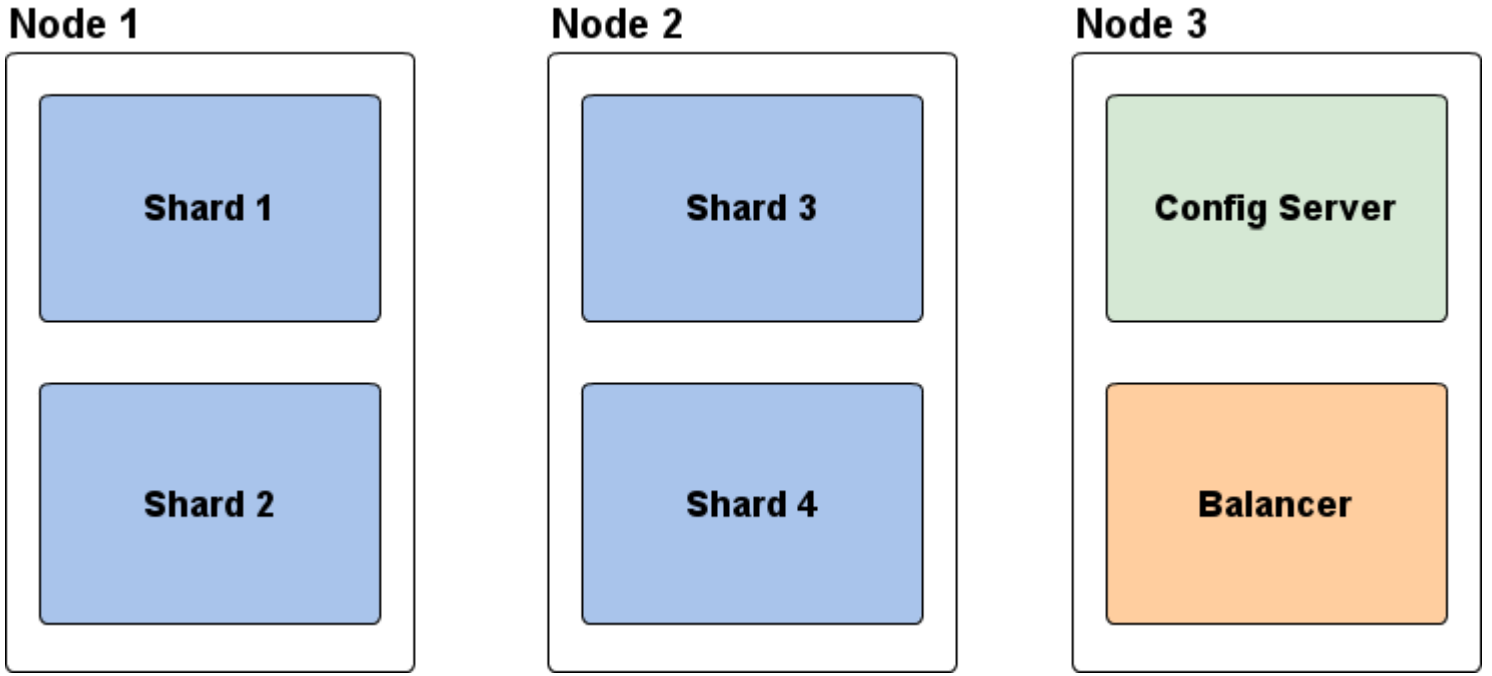
~~2~~ 1 mongos (load balancer) → ~~2~~ 0.5 node

~~3~~ 1 mongod (config servers) → ~~3~~ 0.5 node

Total: ~~11~~ 3 nodes

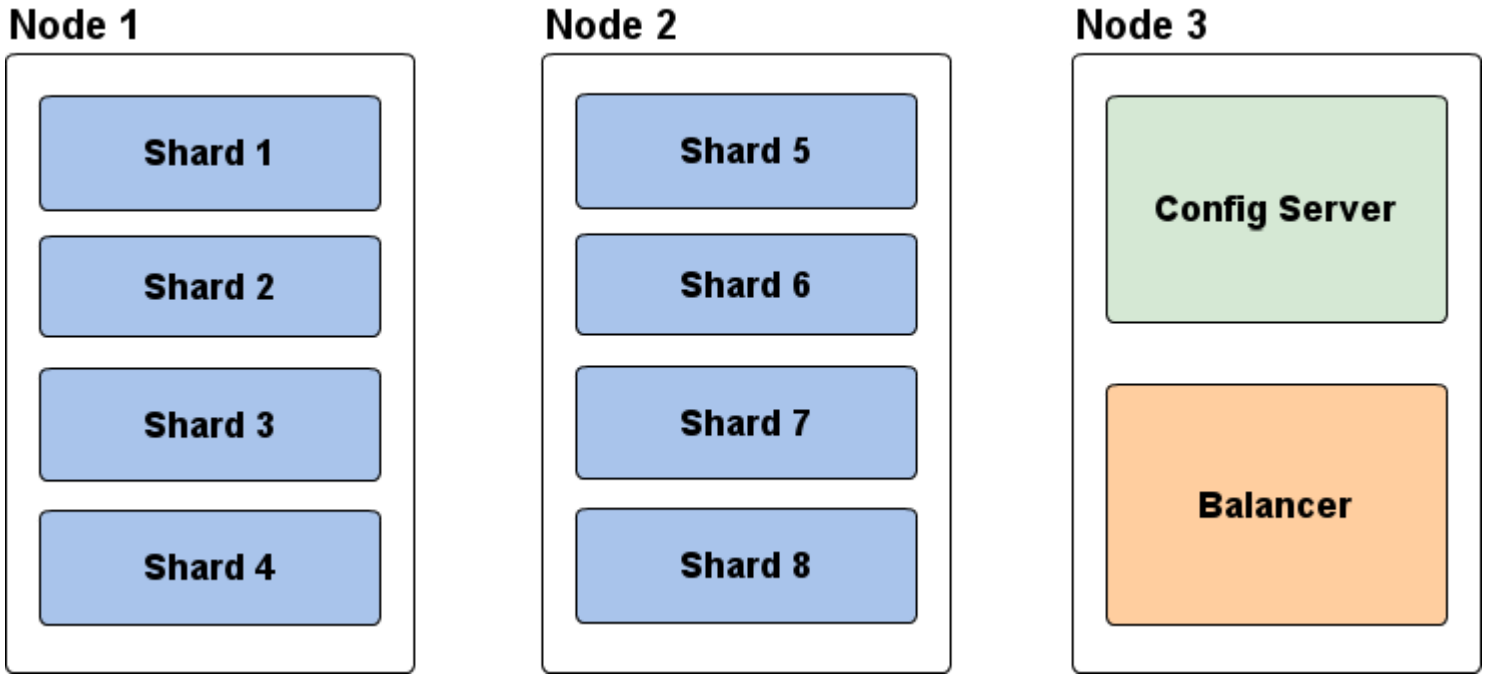
NOT RECOMMENDED
IN PRODUCTION

Which looks something like this:



4 Shard configuration

Or this:



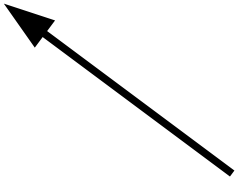
8 Shard configuration

Another problem is with querying in a sharded environment . . .

We *couldn't* perform queries using the Aggregation Framework in the shards, because it was limited by:

- Output at every stage of the aggregation can only contain 16MB

- >10% RAM usage



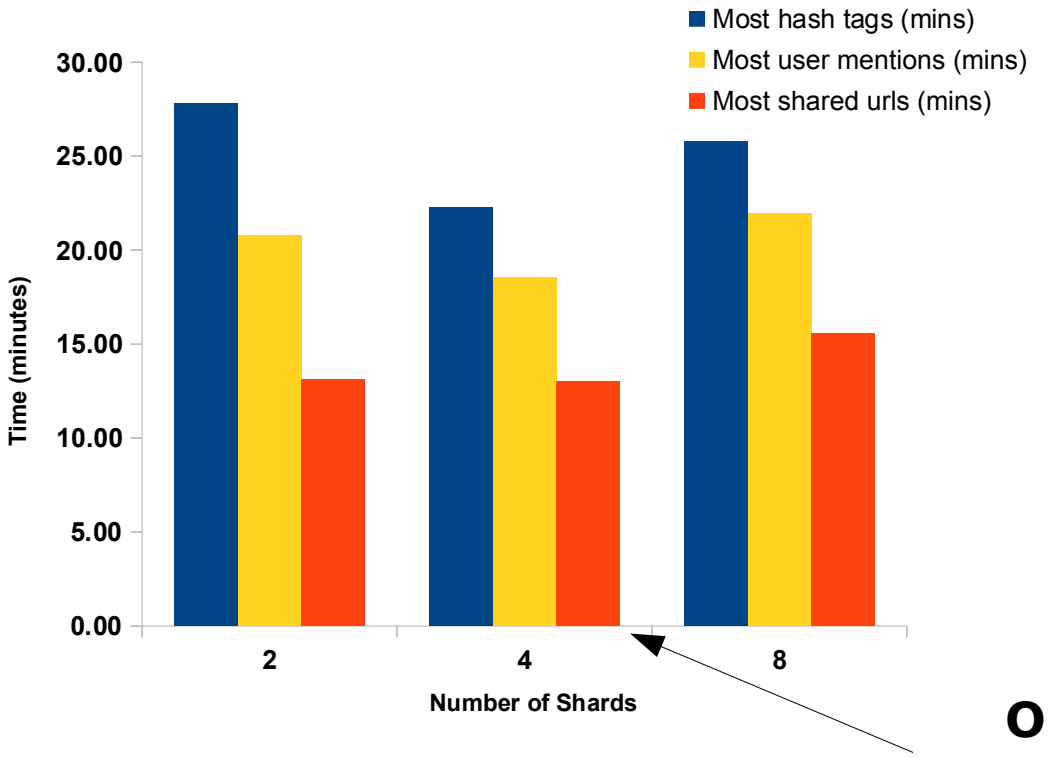
MongoDB would fail if the limits were reached

mongoDB

4 Cores
8GB RAM

Sharded Cluster

Map Reduce: On 3 modest machines



Optimal

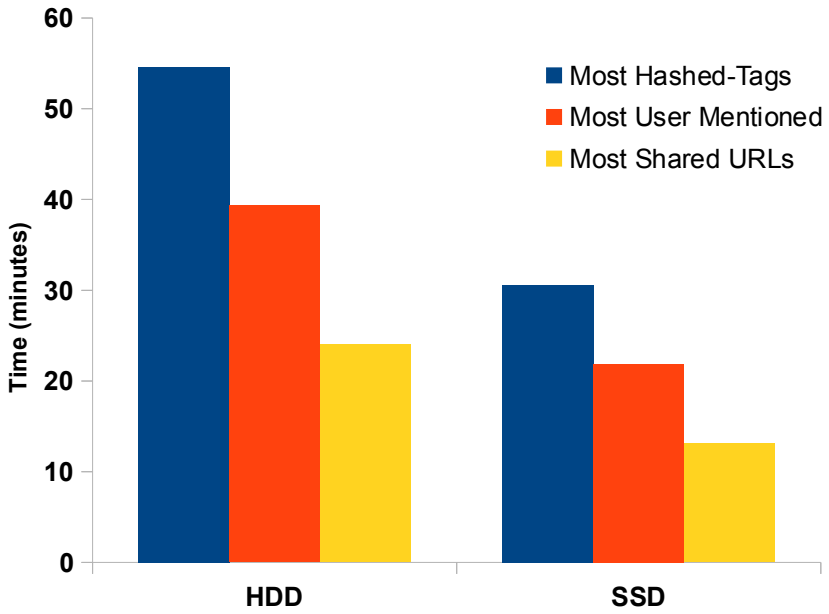
mongoDB

4 Cores
8GB RAM

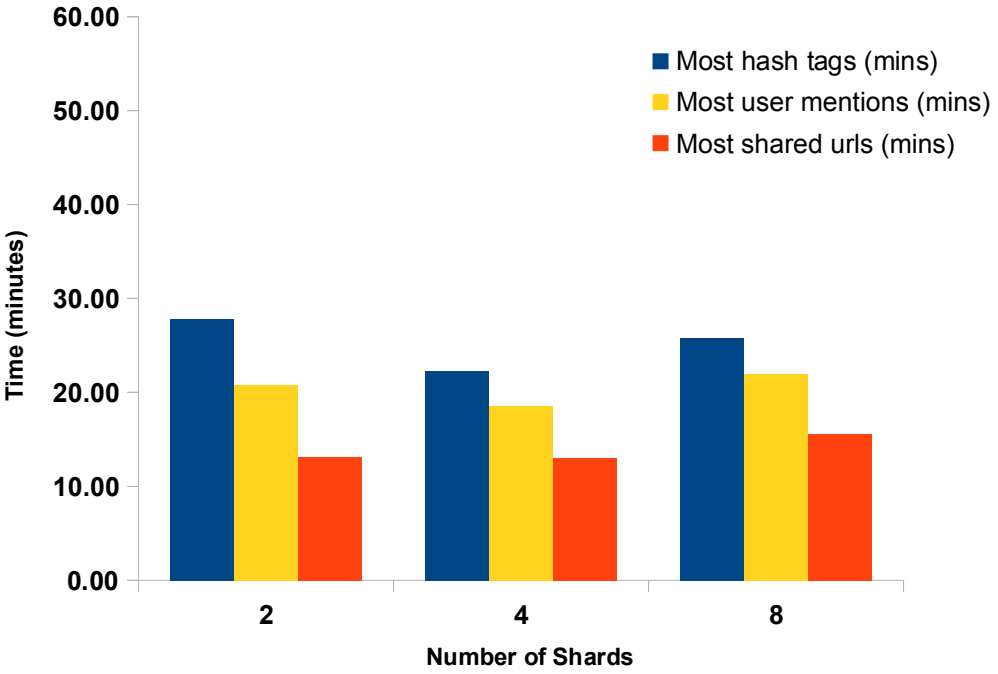
Sharded Cluster

Map Reduce: On 3 modest machines

Single Node



Three Nodes



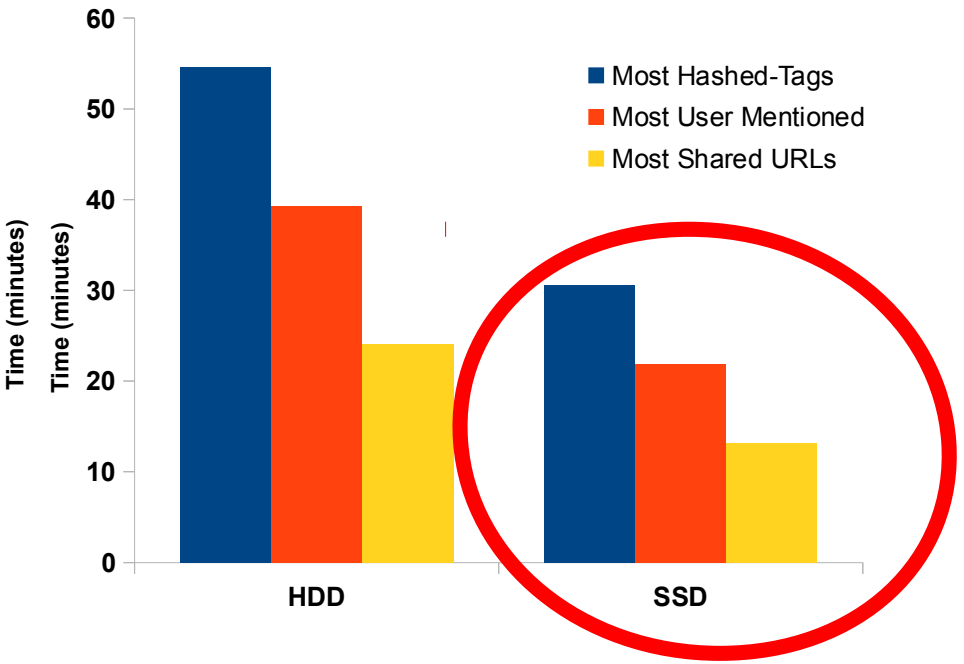
mongoDB

4 Cores
8GB RAM

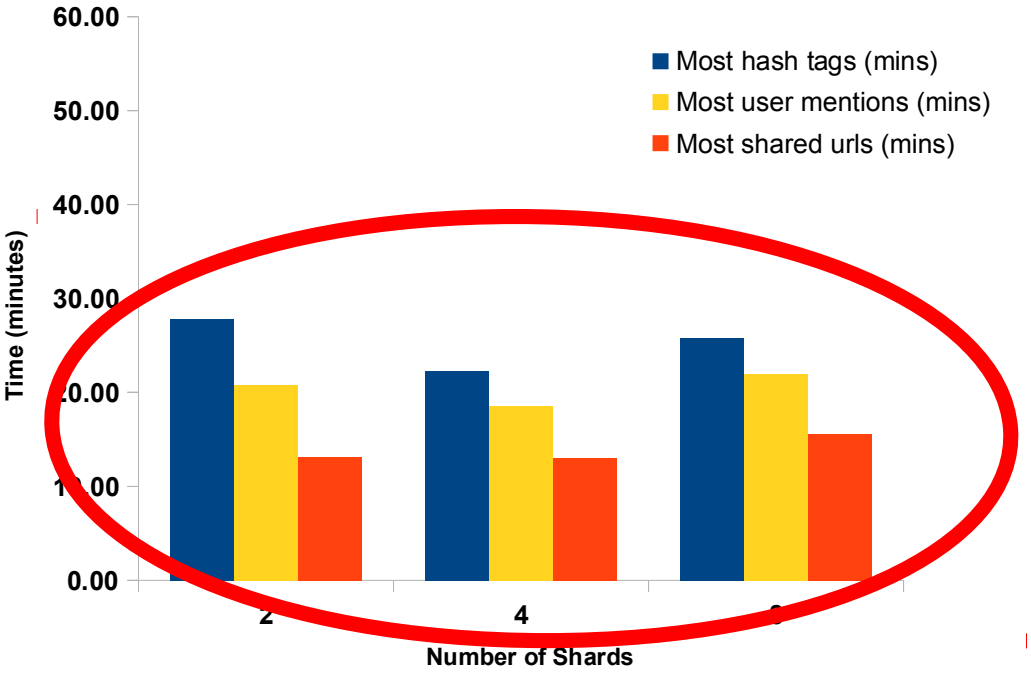
Sharded Cluster

Map Reduce: On 3 modest machines

Single Node



Three Nodes



Performance of Single Node (SSD) is very comparable to 3 nodes (HDD) of different number of shards!

(Single Node with SSD is 2-3 minutes slower than 3 nodes . . . [40 GB corpus])

Perhaps its worth purchasing a good SSD over *investing more nodes* to improve aggregate performance . . .

What happens when we scale vertically

with ONE BIG BOX?

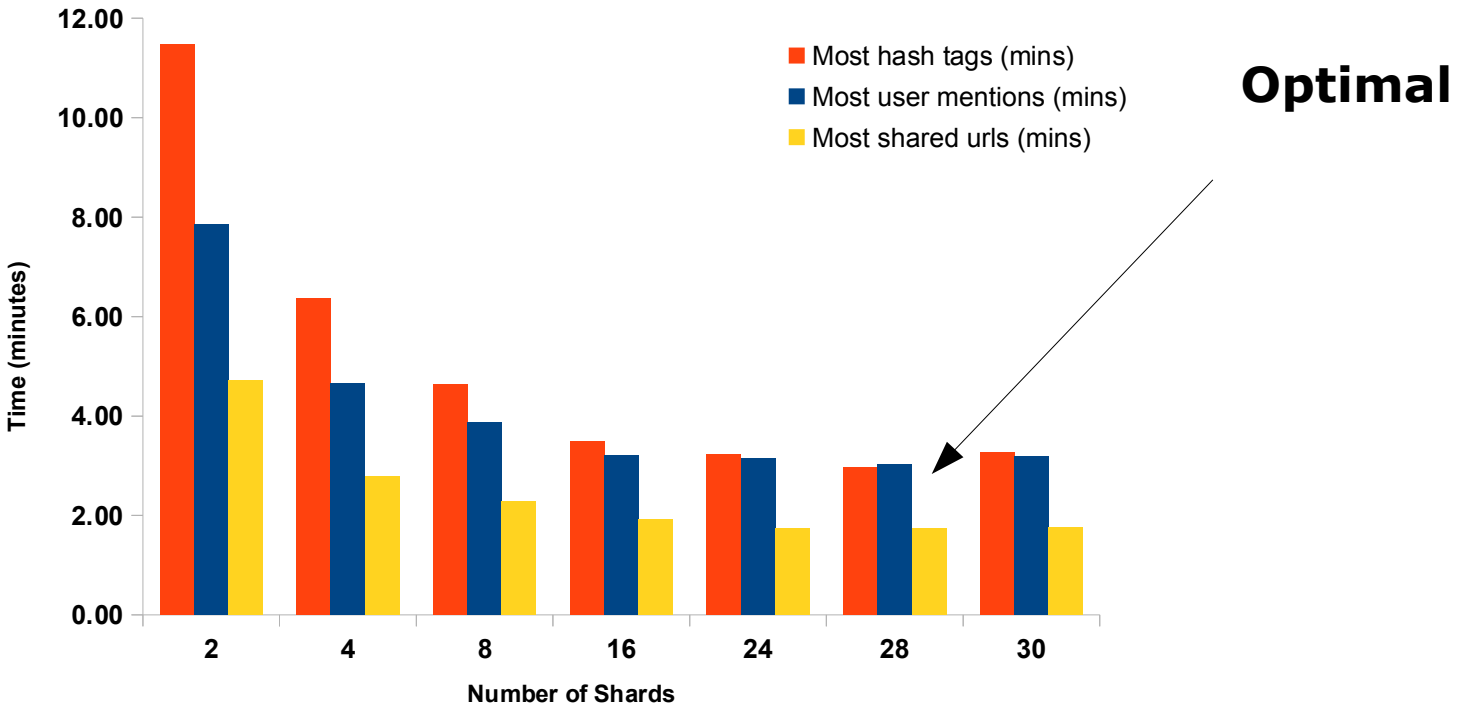
we get --->

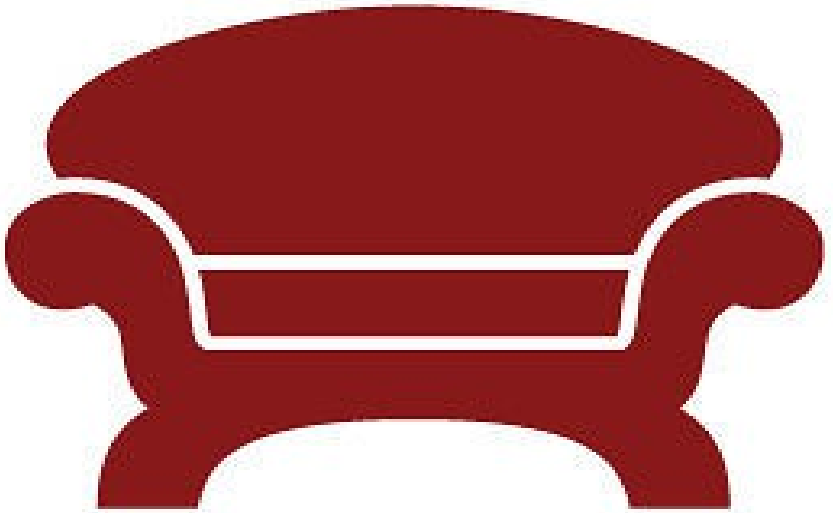
mongoDB

32 Cores
128GB RAM

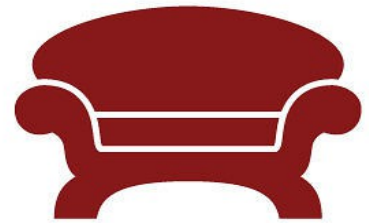
Sharded Cluster

Map Reduce: On a Big Box

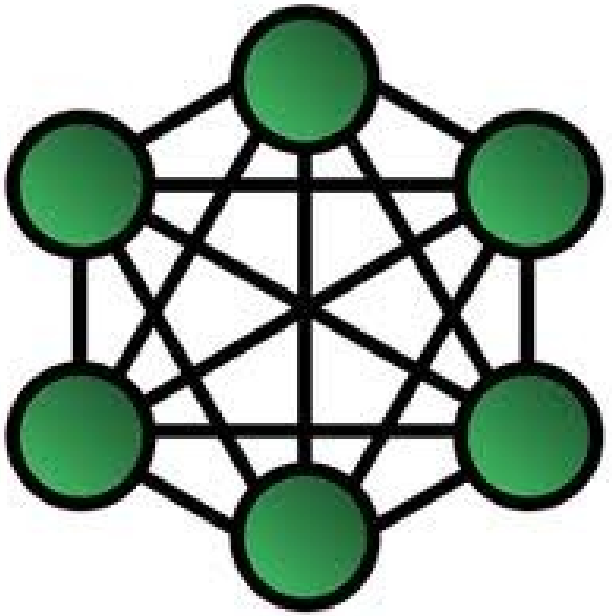




Couchbase



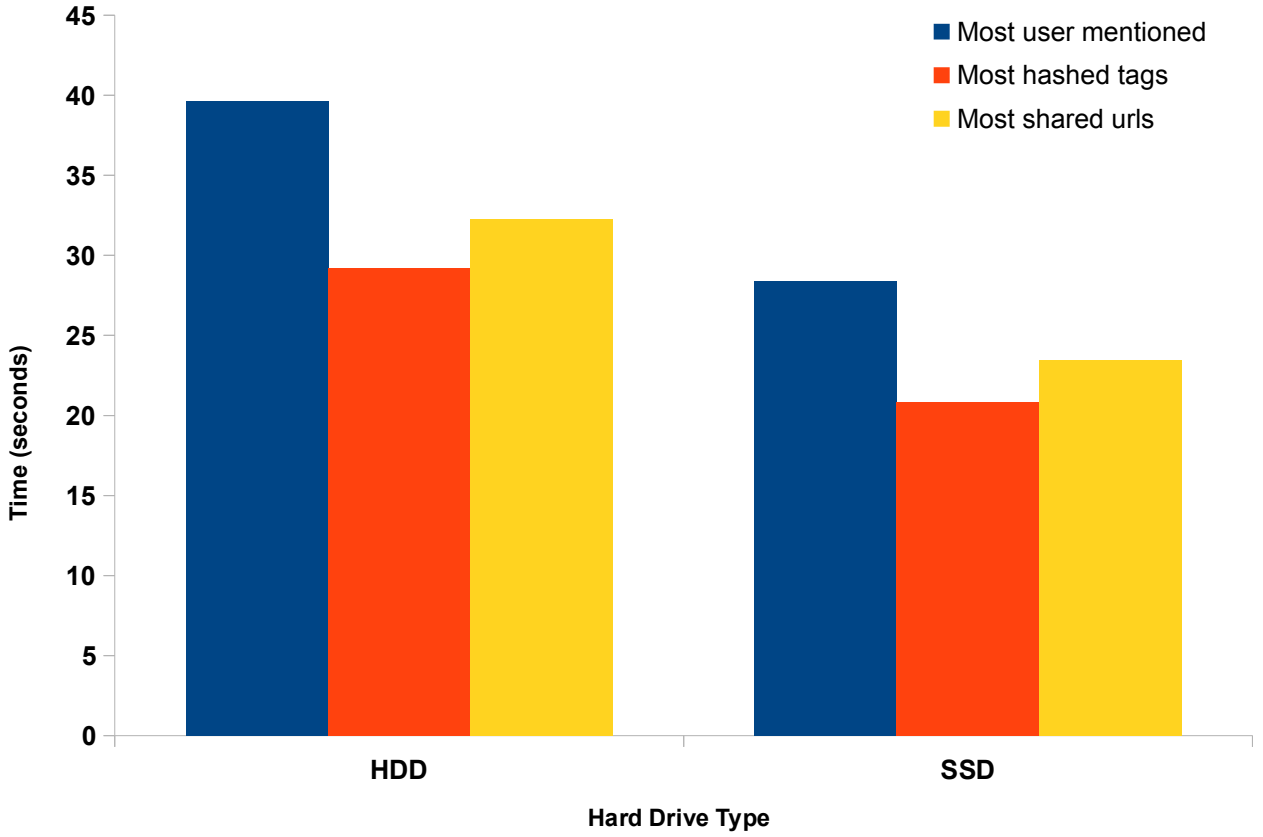
uses a communistic scaling approach



Master-Master Replication

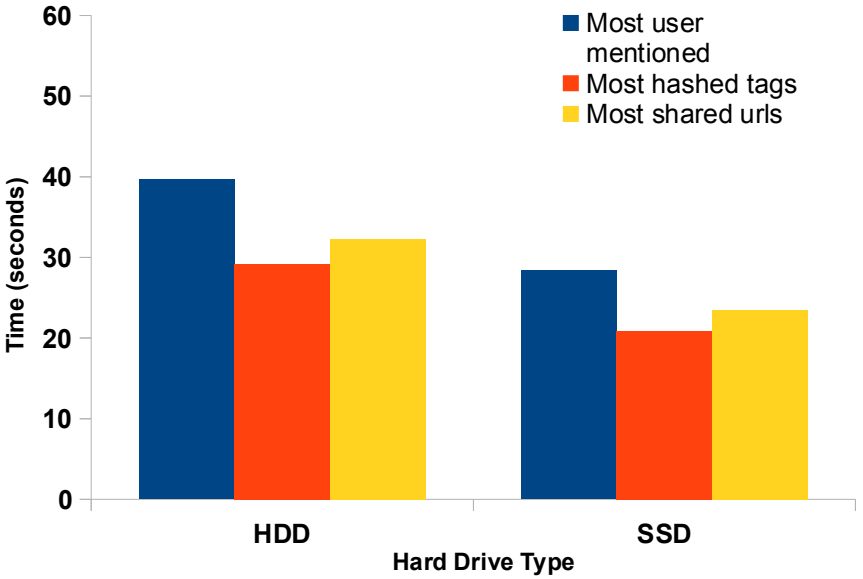

Couchbase

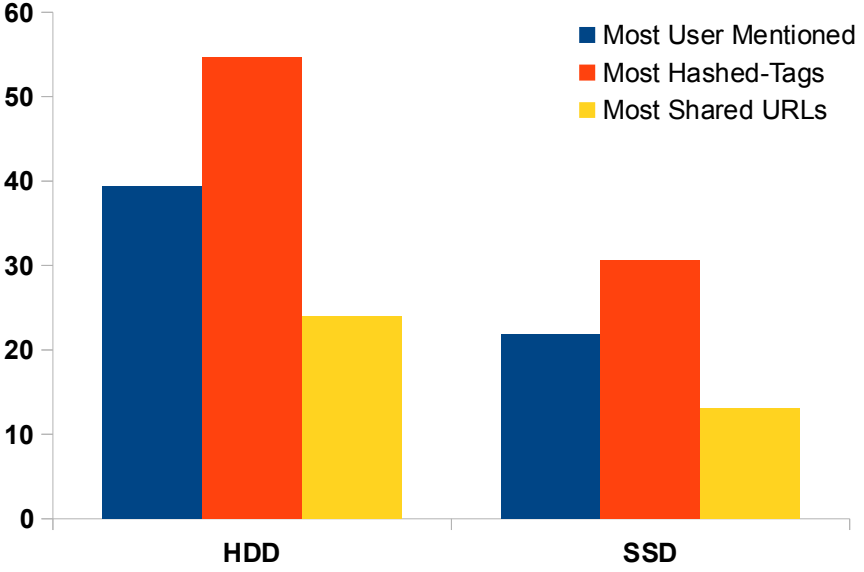
Single Node


45
40
35
30
25
20
15
10
5
0
Time (seconds)
Most user mentioned
Most hashed tags
Most shared urls
HDD
SSD
Hard Drive Type

Single Node
VS

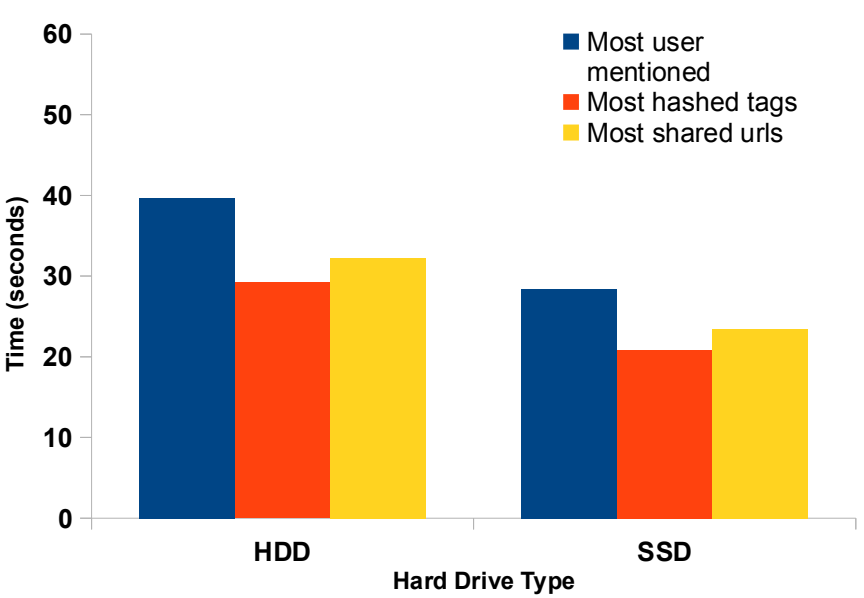
Map Reduce

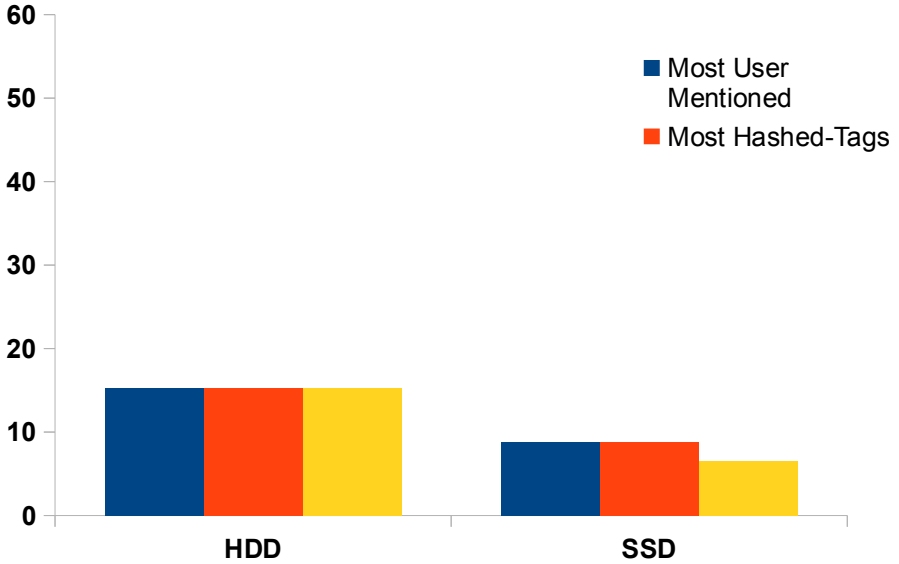




Single Node
VS

Aggregation Framework





On a SINGLE NODE , while

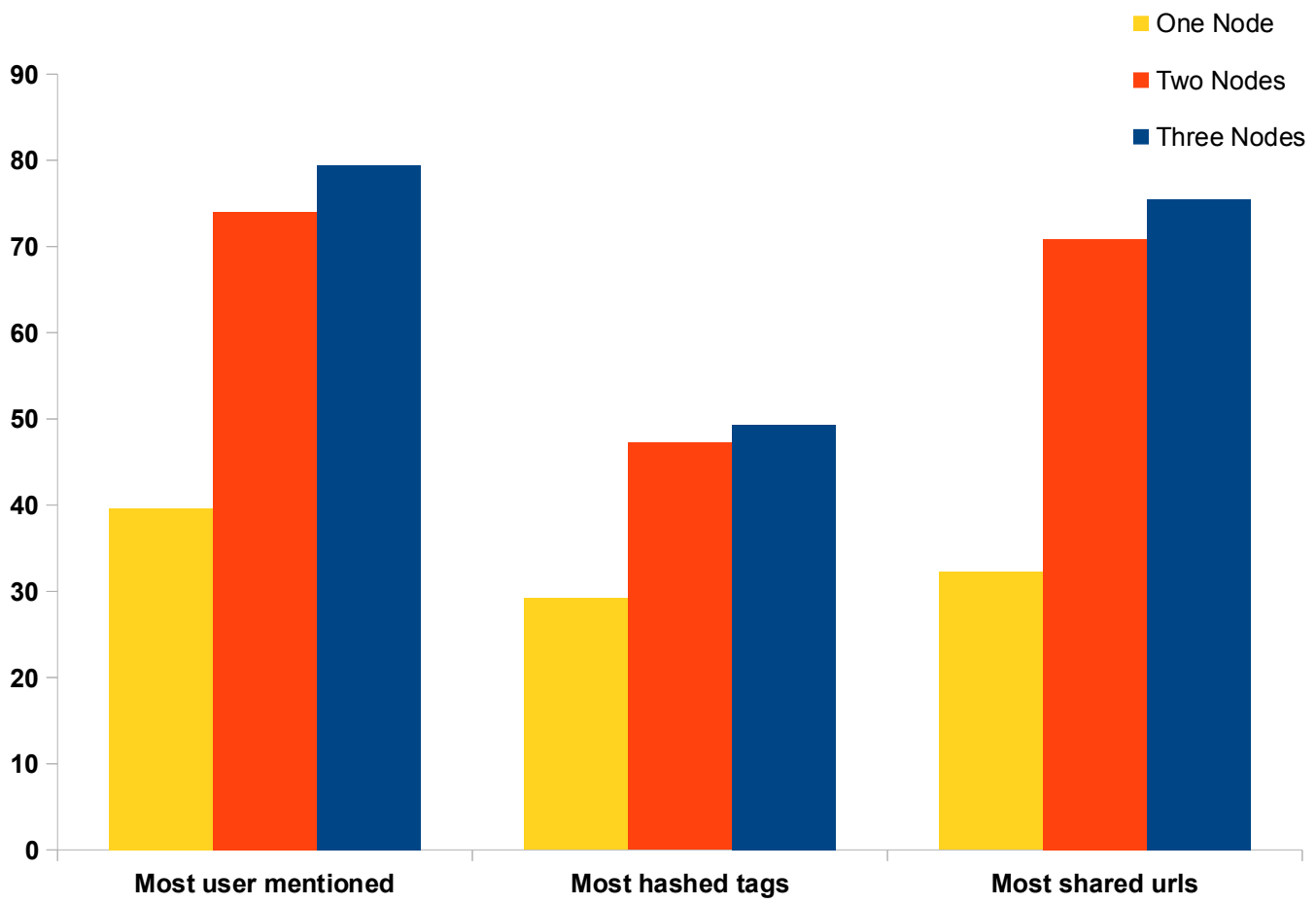
CouchBase
is *FASTER* than
MongoDB's MapReduce

MongoDB's Aggregation Framework
is *FASTER* than
CouchBase

That said adding more Couchbase nodes was disappointing. . .

In short . . .

1 Node VS 2 Nodes VS 3 Nodes



***Adding more nodes* did not improve query performance . . .**

Other users have found similar issues, Couchbase Support says:

"One known issue with the current developer preview of Couchbase is that it *slows down* on *smaller clusters*. The views are optimal on very large clusters." [1]

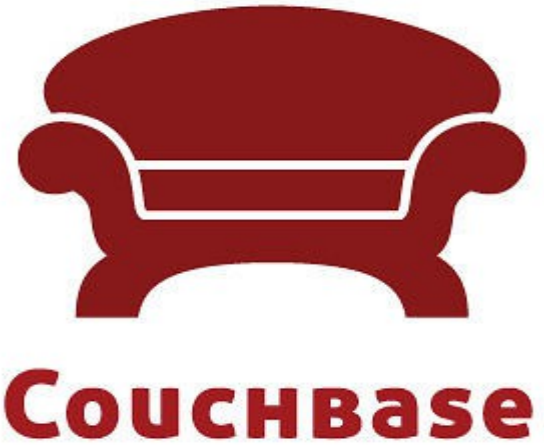
But they did not specify what is "very large"

[1] Couchbase performance - real tests - it's horrible - or am I doing somethin wrong?

That was

November 2011 . . .

Which one?



The obvious answer is

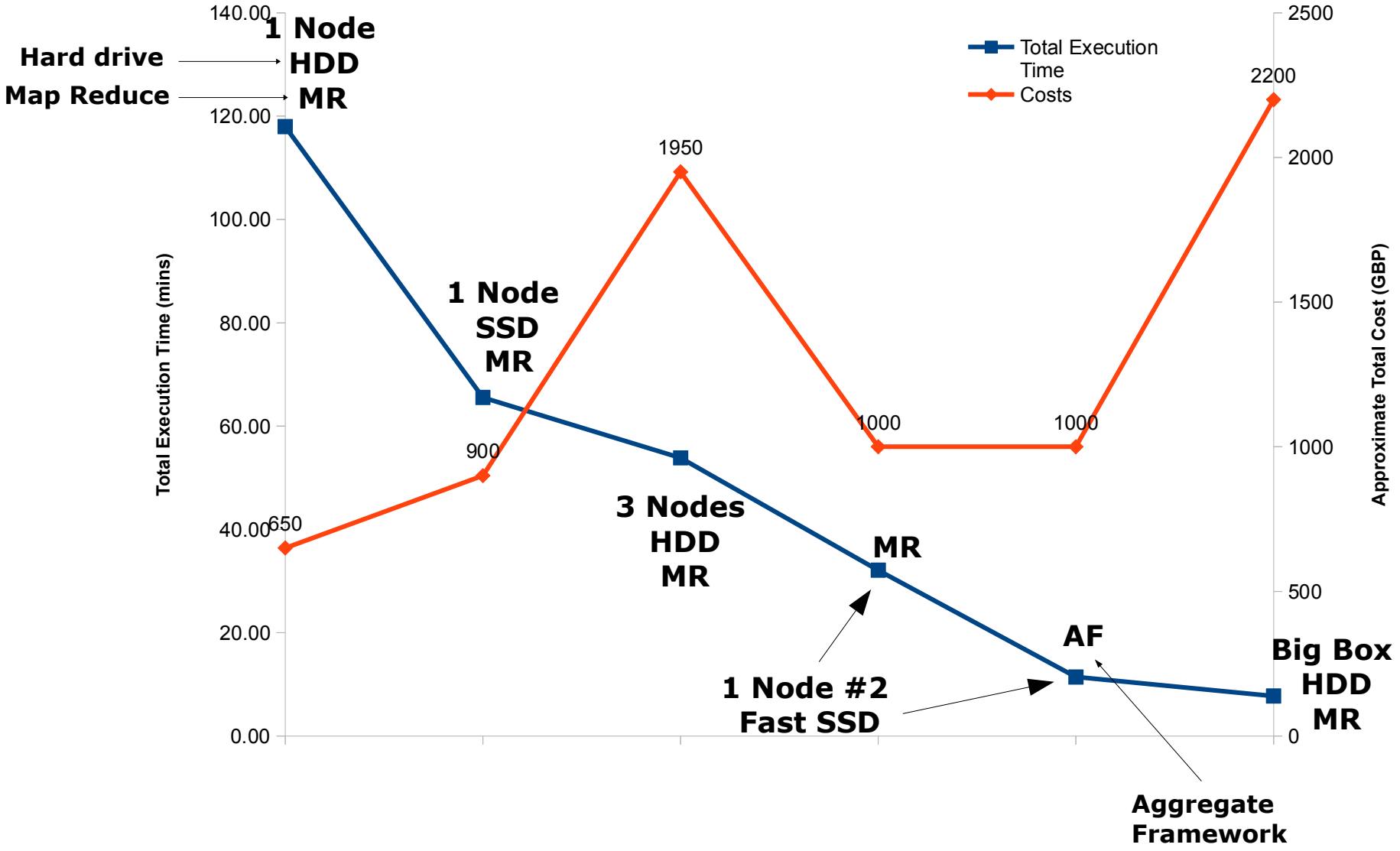
From a *performance standpoint*.

Simple flat file aggregation with *Java* took approx 60 mins

On the same machine, MongoDB took approx

30 mins (MapReduce)
10 mins (Aggregation Framework)

Deployment Options



But!

Has problems with:

- **Full Text Search**
- **Social Network Graphs**
- **Aggregation**

mongoDB's Full Text Search

is ***slow***

1 simple query on a single node, 44GB corpus took approx

10 mins!

Solution:

A *Fast* Full Text Search Engine

The same search took only a fraction of a second

To create a *Social Network Graph* with

Can be *Complex*, since you query multiple times to obtain relationships

Solution:

A *Popular* Graph Database

Where you can obtain a network graph with *VERY FEW* queries

Example:

```
START john=node:node_auto_index(name = 'John')
MATCH john-[:friend]->()-[:friend]->fof
RETURN john, fof
```

The above query fetches all nodes who are friends of friends of “Johns”

With Aggregation

Is *perhaps* not the best, we have yet to play with

Or even

An SQL language to build MapReduce queries

With

Moral of the story

Use the right tools for the right job

Future work

We aim to experiment with:

With possibility in integrating all with