Big Data Engineering

NoSQL databases

Contents

- Why NoSQL?
- ReCAP
- BigTable and Dynamo
- A summary of a few NoSQL databases
  - MongoDB, Cassandra, Couchbase,

Why NoSQL?

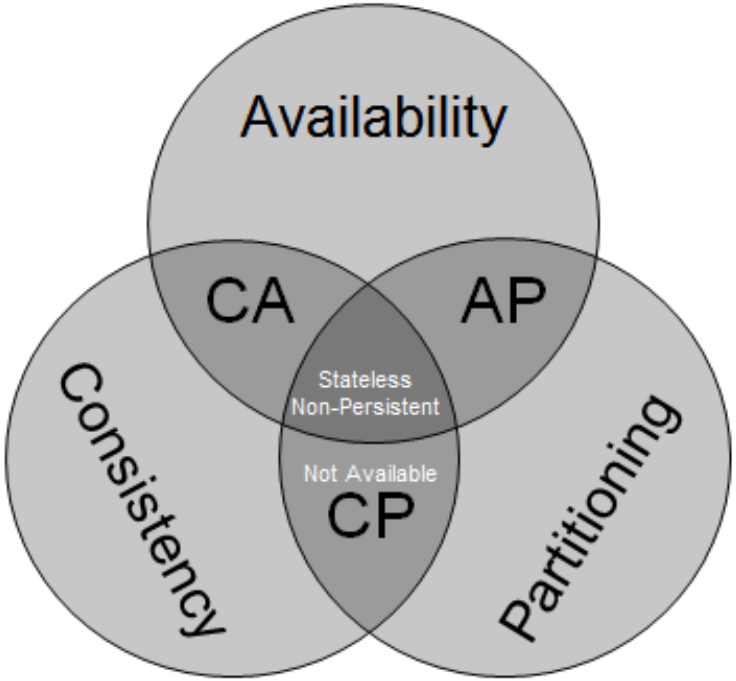
- Availability
  - Need better scaling capabilities
  - Elasticity
- Different schema approaches
  - Graphs, Key Values, Document, Sparse Columns, etc
- More appropriate balance in read/write performance
- Better integration with REST/SOA/Cloud

NoSQL history

- Not just a recent thing ☺
- IBM IMS (Information Management System)
  - Launched in 1968
  - Used to store the bill of materials for the Saturn V rocket
  - Hierarchical model
- Still in widespread use today

ReCAP

- You can have 2 out of three:
  - Consistent
    - ACID
  - Available
    - HA / Accessible 24x7
  - Partitioned
    - Able to split into different datacentres
    - Survive network down

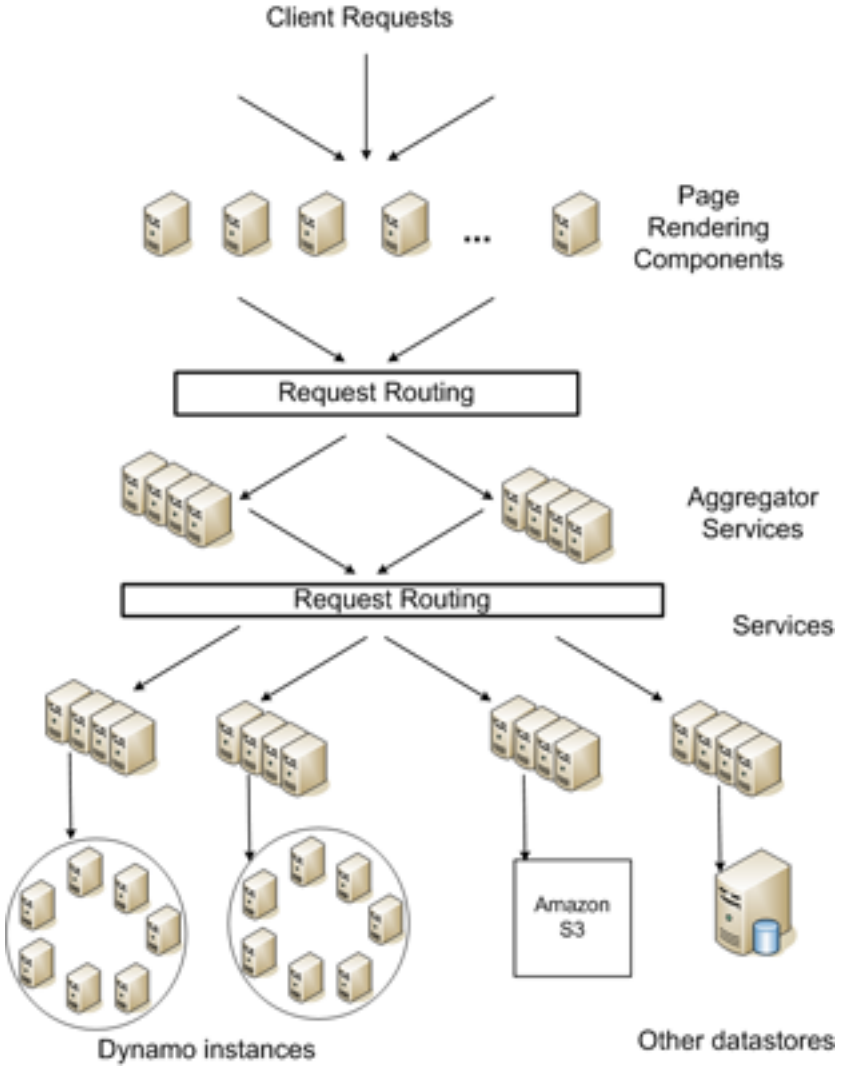


http://blog.mccrory.me/2010/11/03/cap-theorem-and-the-clouds/

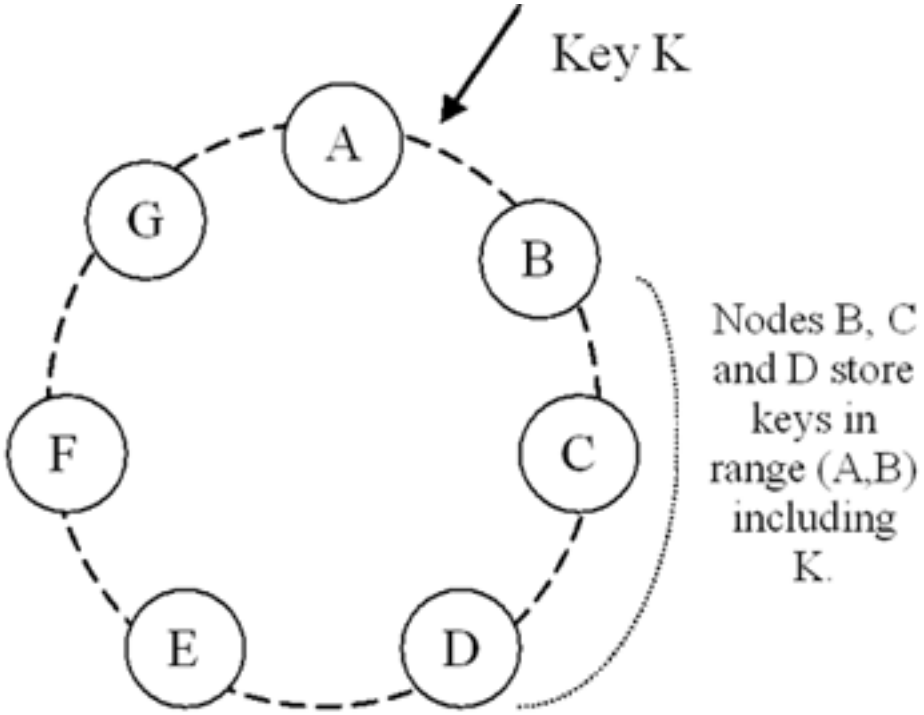
NoSQL parents

- Amazon Dynamo
  - Eventually consisten
- Google BigTable
  - Supporting very large rows
- LDM
  - Graph database

Dynamo

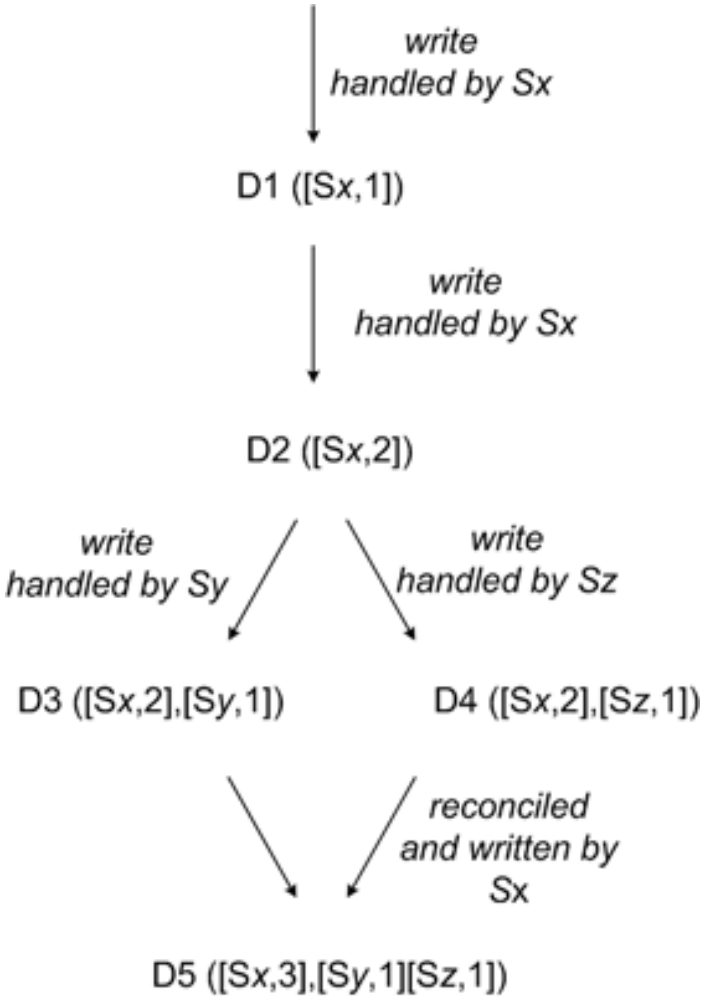


Dynamo Model



http://www.allthingsdistributed.com/2007/10/amazons_dynamo.html

Reconciliation / Eventual Consistency



http://www.allthingsdistributed.com/2007/10/amazons_dynamo.html

Dynamo Techniques

Problem	Technique	Advantage
Partitioning	Consistent Hashing	Incremental Scalability
High Availability for writes	Vector clocks with reconciliation during reads	Version size is decoupled from update rates.
Handling temporary failures	Sloppy Quorum and hinted handoff	Provides high availability and durability guarantee when some of the replicas are not available.
Recovering from permanent failures	Anti-entropy using Merkle trees	Synchronizes divergent replicas in the background.
Membership and failure detection	Gossip-based membership protocol and failure detection.	Preserves symmetry and avoids having a centralized registry for storing membership and node liveness information.

Google BigTable

- Optimized to support very large data
  - Not just many rows, but rows that cannot fit into the memory of a single server
  - Column Families allow each row to live across servers
- This table dates back to 2005

Project name	Table size (TB)	Compression ratio	# Cells (billions)	# Column Families	# Locality Groups	% in memory	Latency-sensitive?
Crawl	800	11%	1000	16	8	0%	No
Crawl	50	33%	200	2	2	0%	No
Google Analytics	20	29%	10	1	1	0%	Yes
Google Analytics	200	14%	80	1	1	0%	Yes
Google Base	2	31%	10	29	3	15%	Yes
Google Earth	0.5	64%	8	7	2	33%	Yes
Google Earth	70	–	9	8	3	0%	No
Orkut	9	–	0.9	8	5	1%	Yes
Personalized Search	4	47%	6	93	11	5%	Yes

Current NoSQL Databases

- Too many to list!
- Popular databases include:
  - MongoDB
  - Couchbase
  - Apache Cassandra
  - Apache HBase
  - Voldemort
  - Redis
  - Riak
  - Etc, etc

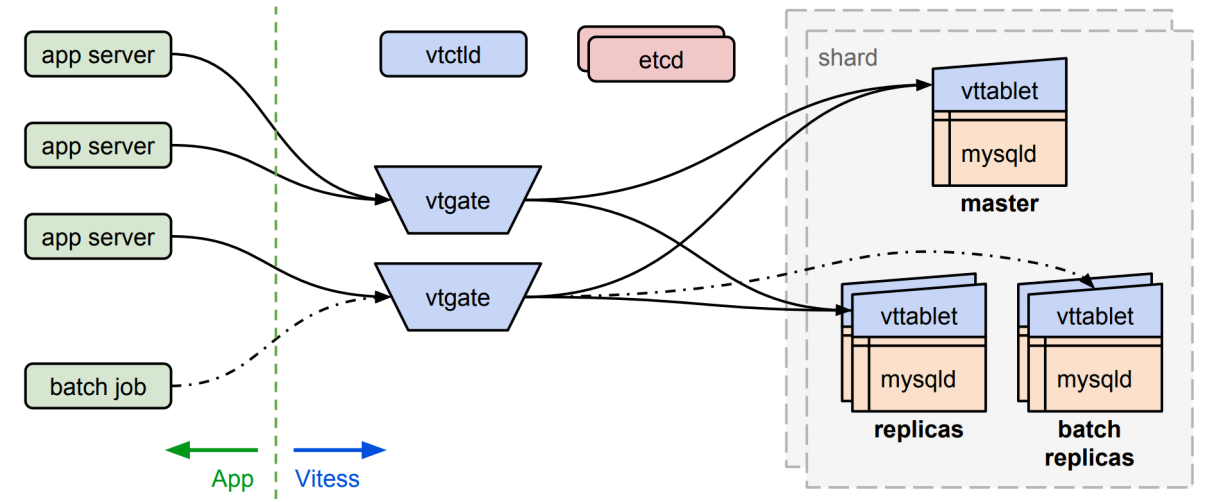
“NewSQL”

- ACID databases that aim to provide HA and Partition safety
  - VoltDB
  - NuoDB
  - Google Spanner
  - MemSQL
  - SAP HANA
- Also there are some backend engines for MySQL that aim to provide this:
  - MySQL Cluster
  - TokuDB

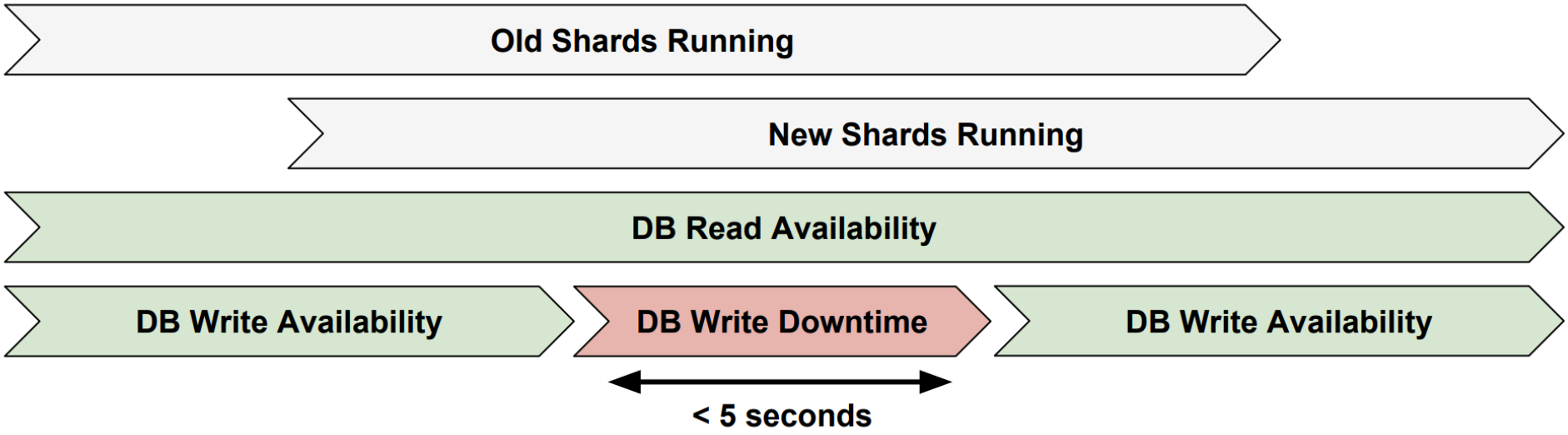
Vitess

https://github.com/youtube/vitess

Components



https://freo.me/vitess-pres



In Memory Databases

- Memory is relatively much cheaper than it used to be
- Uses snapshots or transaction logs to ensure durability
- *Some NoSQL, some NewSQL*
  - SAP Hana
  - Redis
  - VoltDB
  - MemSQL
  - Apache Geode

Top ten databases

283 systems in ranking, November 2015

Rank Nov 2015	Rank Oct 2015	Rank Nov 2014	DBMS	Database Model	Score Nov 2015	Score Oct 2015	Score Nov 2014
1.	1.	1.	Oracle	Relational DBMS	1480.95	+13.99	+28.82
2.	2.	2.	MySQL	Relational DBMS	1286.84	+7.88	+7.77
3.	3.	3.	Microsoft SQL Server	Relational DBMS	1122.33	-0.90	-97.87
4.	4.	↑ 5.	MongoDB ⊞	Document store	304.61	+11.34	+59.87
5.	5.	↓ 4.	PostgreSQL	Relational DBMS	285.69	+3.56	+28.33
6.	6.	6.	DB2	Relational DBMS	202.52	-4.28	-3.71
7.	7.	7.	Microsoft Access	Relational DBMS	140.96	-0.87	+2.12
8.	8.	↑ 9.	Cassandra ⊞	Wide column store	132.92	+3.91	+40.93
9.	9.	↓ 8.	SQLite	Relational DBMS	103.45	+0.78	+8.17
10.	10.	↑ 11.	Redis ⊞	Key-value store	102.41	+3.61	+20.06

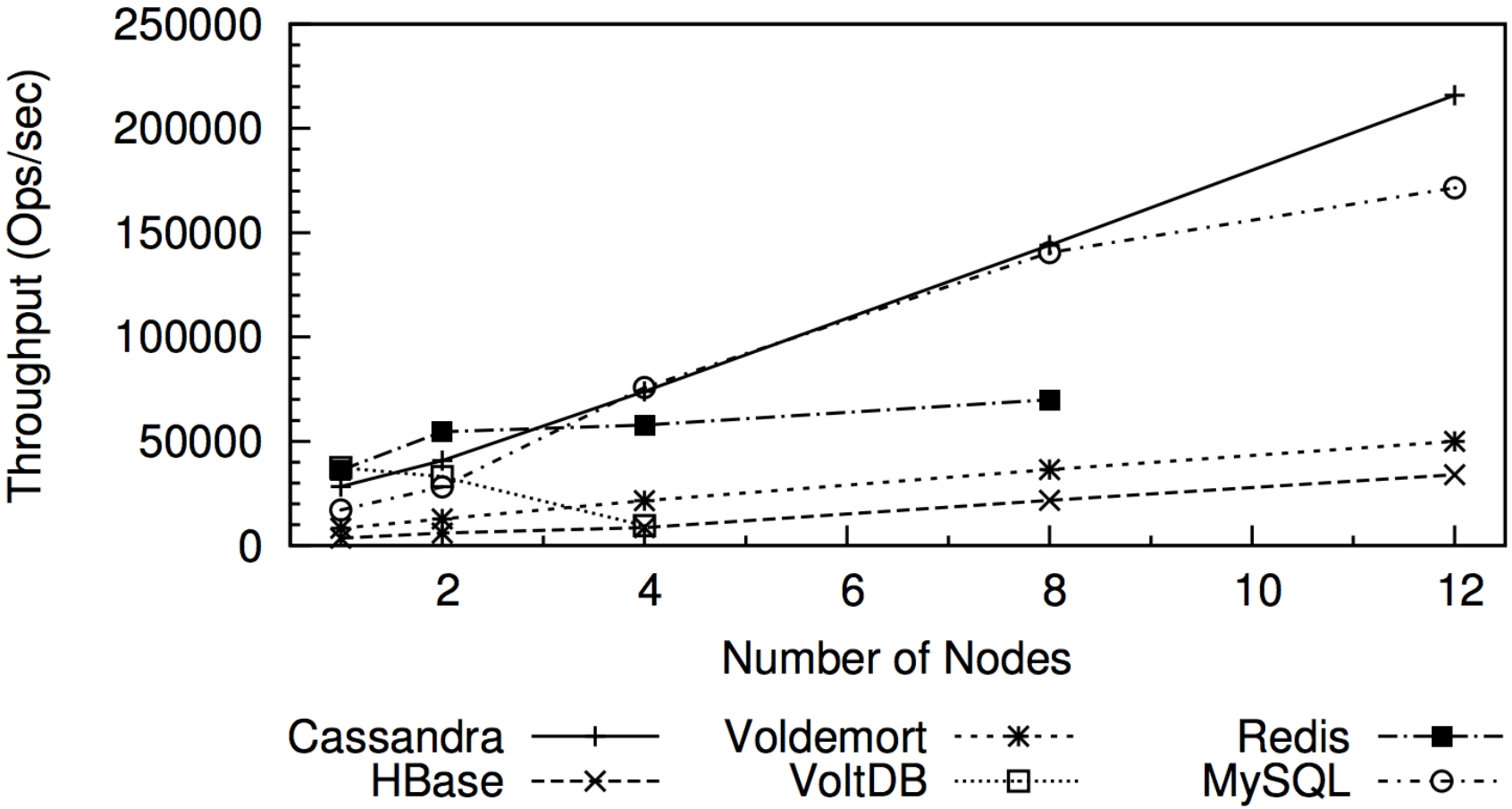
http://db-engines.com/en/ranking

Next 20

11.	11.	↓ 10.	SAP Adaptive Server	Relational DBMS	83.71	-1.93	-0.91
12.	12.	12.	Solr	Search engine	79.77	+0.71	+2.96
13.	13.	13.	Teradata	Relational DBMS	77.08	+3.64	+9.85
14.	14.	↑ 16.	Elasticsearch	Search engine	74.77	+4.55	+31.71
15.	15.	15.	HBase	Wide column store	56.46	-0.78	+9.49
16.	16.	↑ 17.	Hive	Relational DBMS	54.91	+1.35	+18.30
17.	17.	↓ 14.	FileMaker	Relational DBMS	51.73	+1.95	+0.39
18.	18.	↑ 20.	Splunk	Search engine	44.61	+1.11	+12.44
19.	19.	↑ 21.	SAP HANA	Relational DBMS	39.62	+0.52	+11.26
20.	20.	↓ 18.	Informix	Relational DBMS	38.46	+0.12	+2.88
21.	21.	↑ 23.	Neo4j	Graph DBMS	34.04	+0.63	+9.39
22.	22.	↓ 19.	Memcached	Key-value store	32.39	+0.82	-0.20
23.	↑ 25.	↑ 27.	MariaDB	Relational DBMS	26.64	+2.01	+9.35
24.	↓ 23.	↓ 22.	CouchDB	Document store	26.37	-0.49	+0.57
25.	↓ 24.	↓ 24.	Couchbase	Document store	25.82	-0.37	+4.33
26.	26.	26.	Firebird	Relational DBMS	22.56	-0.33	+4.95
27.	↑ 28.	↓ 25.	Netezza	Relational DBMS	21.84	+0.60	+3.76
28.	↓ 27.	↑ 31.	Amazon DynamoDB	Multi-model	21.75	+0.42	+9.43
29.	↑ 30.	↓ 28.	Microsoft Azure SQL Database	Relational DBMS	19.46	+0.62	+4.85
30.	↓ 29.	↓ 29.	Vertica	Relational DBMS	19.41	+0.45	+5.46

Performance (2012)

50%/50% reads/writes

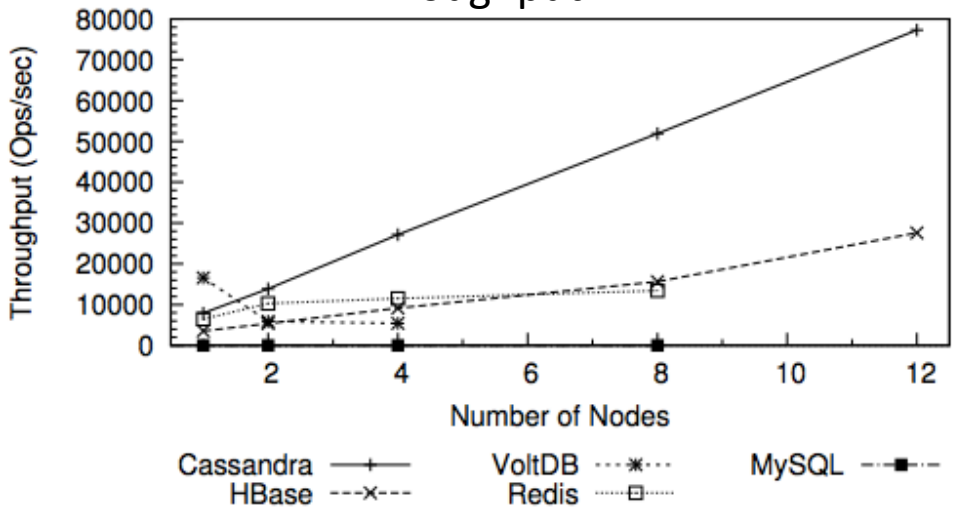


Rabl, Tilmann, et al. "Solving big data challenges for enterprise application performance management." Proceedings of the VLDB Endowment 5.12 (2012): 1724-1735.

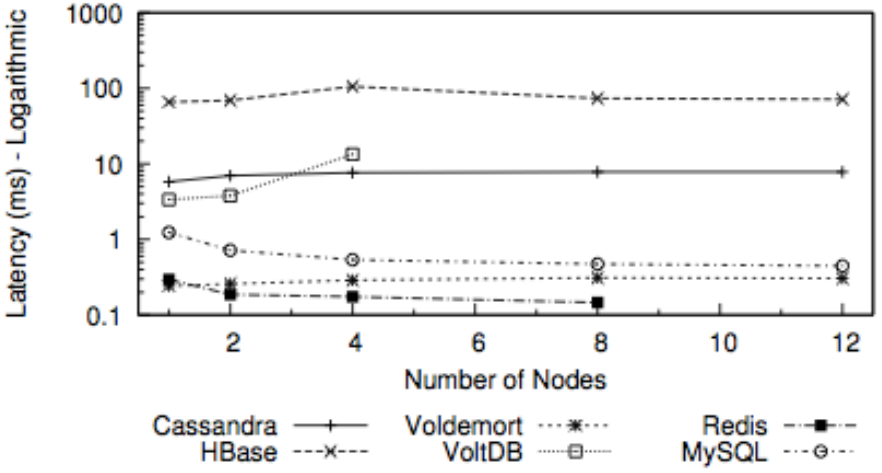
More performance (2012)

Read/Scan/Write workload

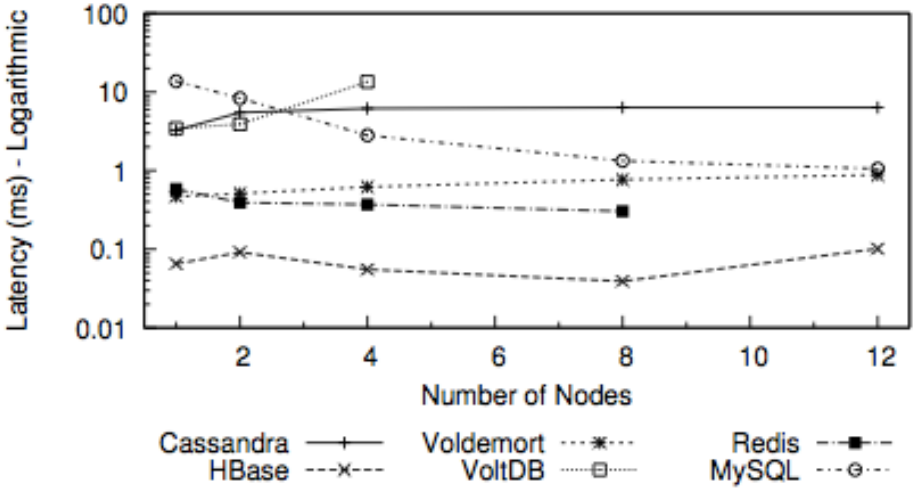
Throughput



Read Latency



Write Latency



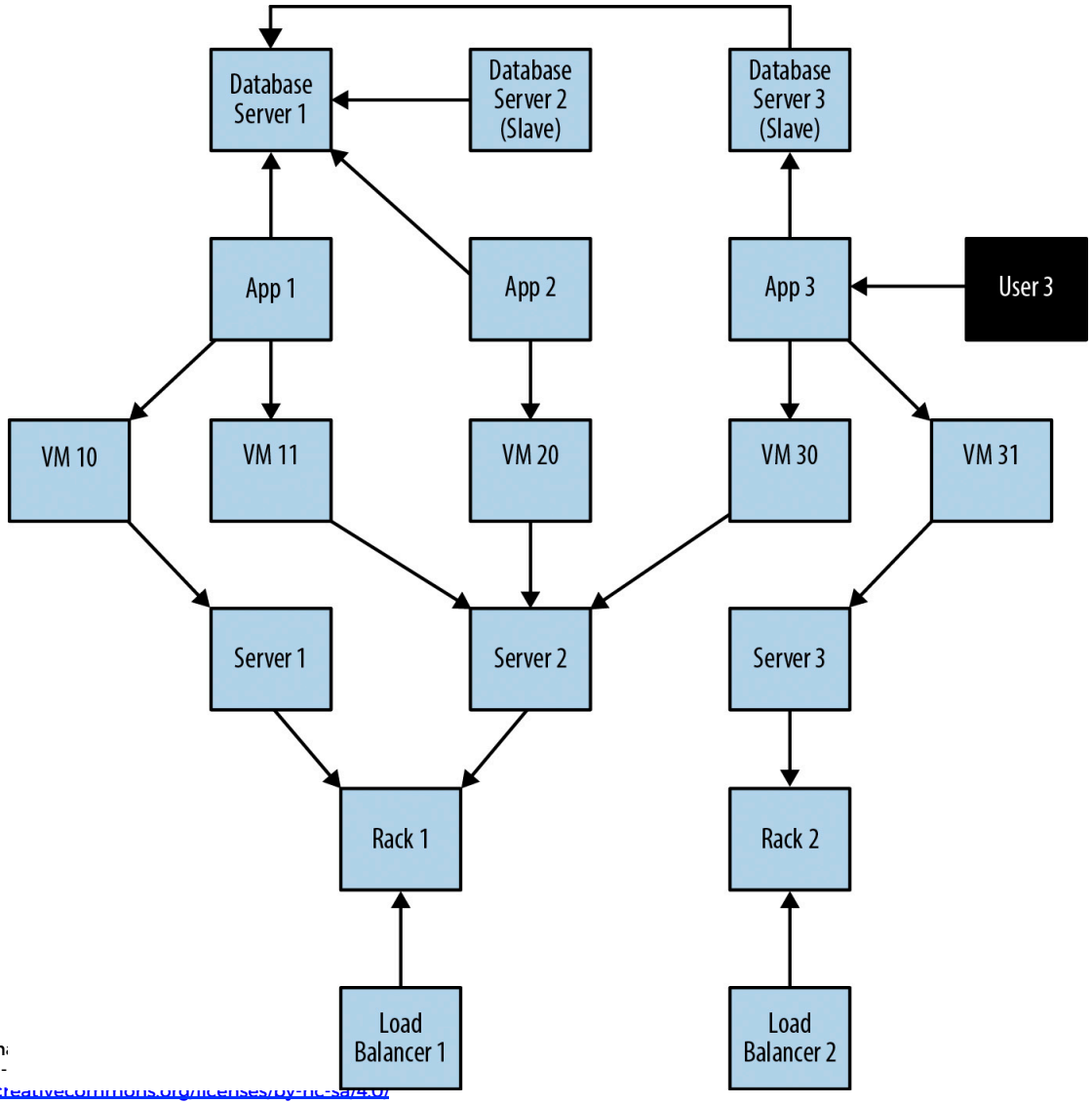
Summary of Performance benchmark (2012)

- **Cassandra** had the best throughput but high latency
- **Voldemort** had the best and most stable latency but lower throughput
- **HBase** had low performance per node but scaled well
  - Low write latency
- **Redis, MySQL and VoltDB** did not scale as well in multi-node setups

Key Value databases

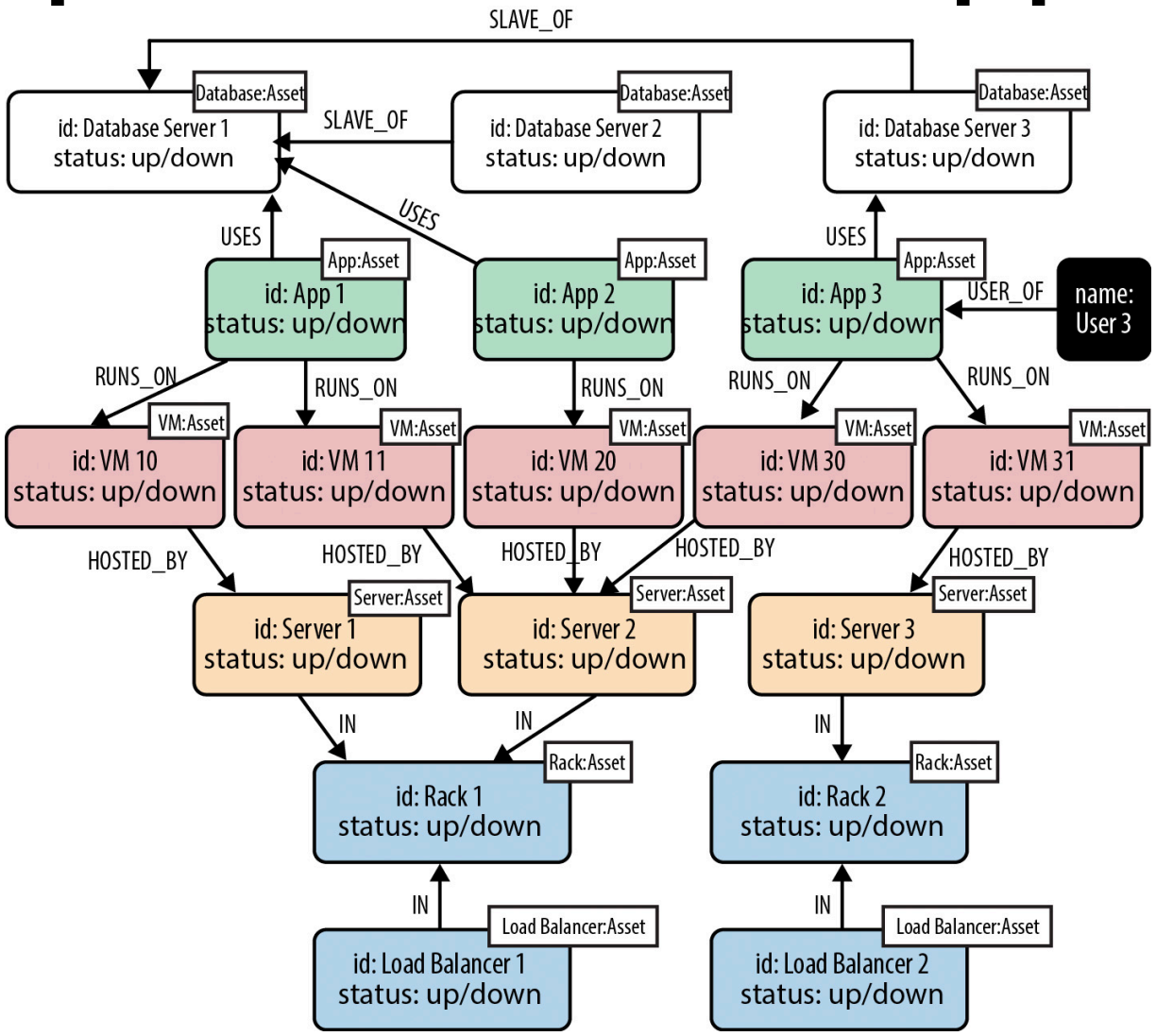
- A persistent associative array or dictionary
- Simple access and fits well with programming models (especially MR)
- Indexing on other data is not often possible and can be slow

Graph Databases



Source: neo4j

Graph Database mapping



Source: neo4j

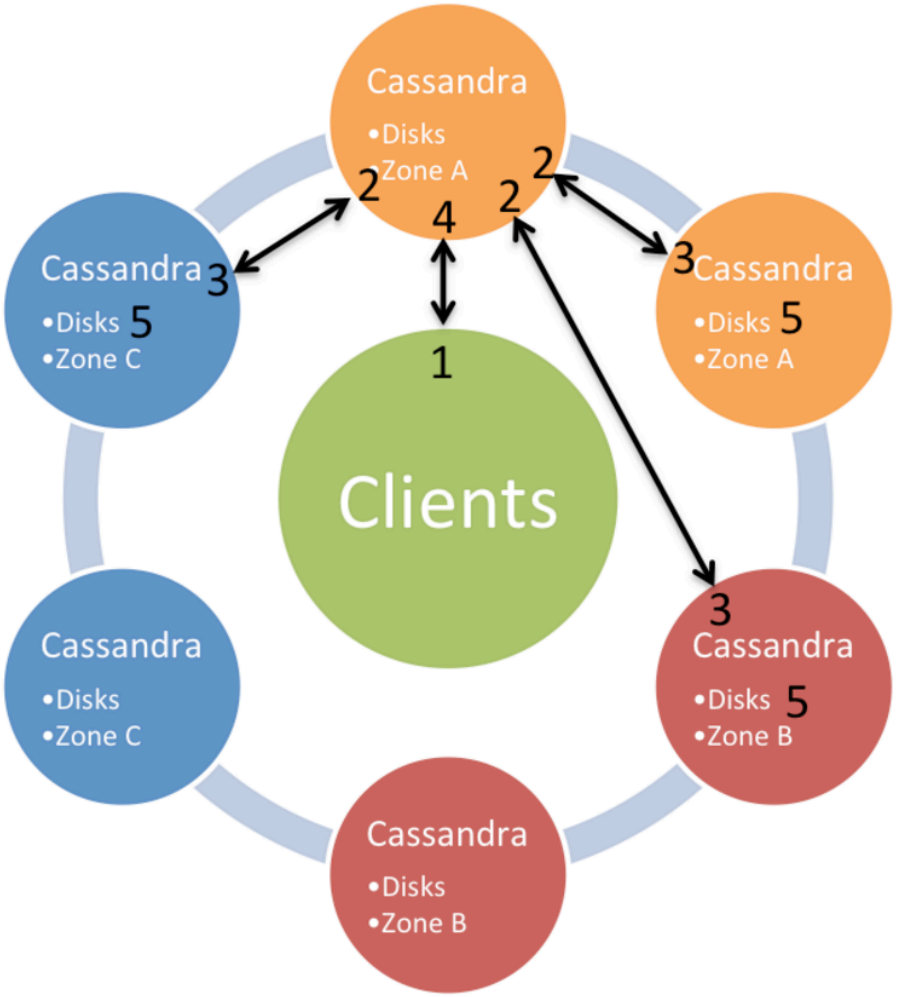
Apache Cassandra

- Masterless / Symmetric
  - Every node is equal and you can write to any node as well as read
- Shared Nothing architecture
  - Each server has its own disk
- Based on Dynamo
  - for automatic sharding and eventual consistency
- And BigTable
  - For “Column Families”
- Donated to Apache by Facebook
  - Now mostly developed by DataStax

Cassandra Write Model

Single Datacentre

1. Client Writes to any Cassandra Node
2. Coordinator Node replicates to nodes and Zones
3. Nodes return ack to coordinator
4. Coordinator returns ack to client
5. Data written to internal commit log disk



If a node goes offline, hinted handoff completes the write when the node comes back up.

Requests can choose to wait for one node, a quorum, or all nodes to ack the write

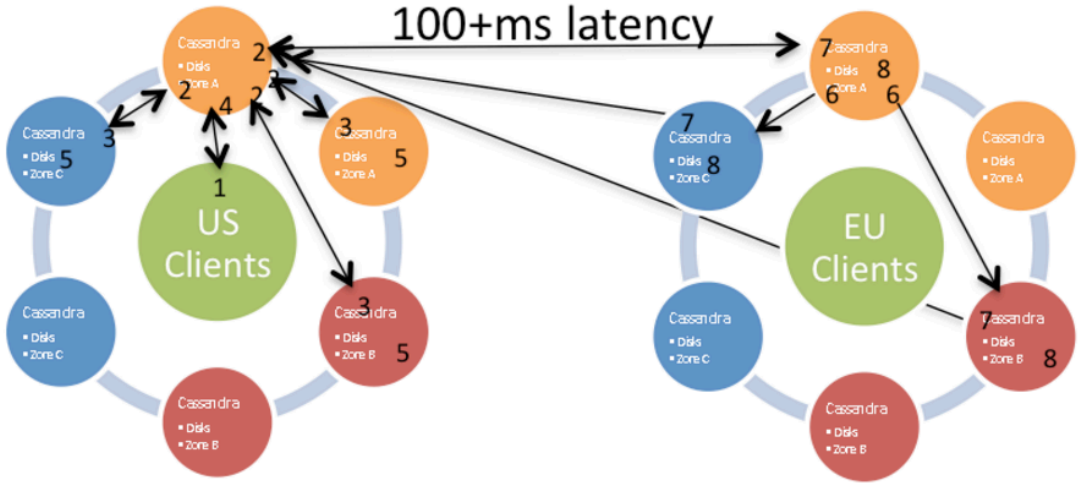
SSTable disk writes and compactions occur asynchronously

Source: Netflix

Multi Datacentre Writes

1. Client Writes to any Cassandra Node
2. Coordinator node replicates to other nodes Zones and regions
3. Local write acks returned to coordinator
4. Client gets ack when 2 of 3 local nodes are committed
5. Data written to internal commit log disks
6. When data arrives, remote node replicates data
7. Ack direct to source region coordinator
8. Remote copies written to commit log disks

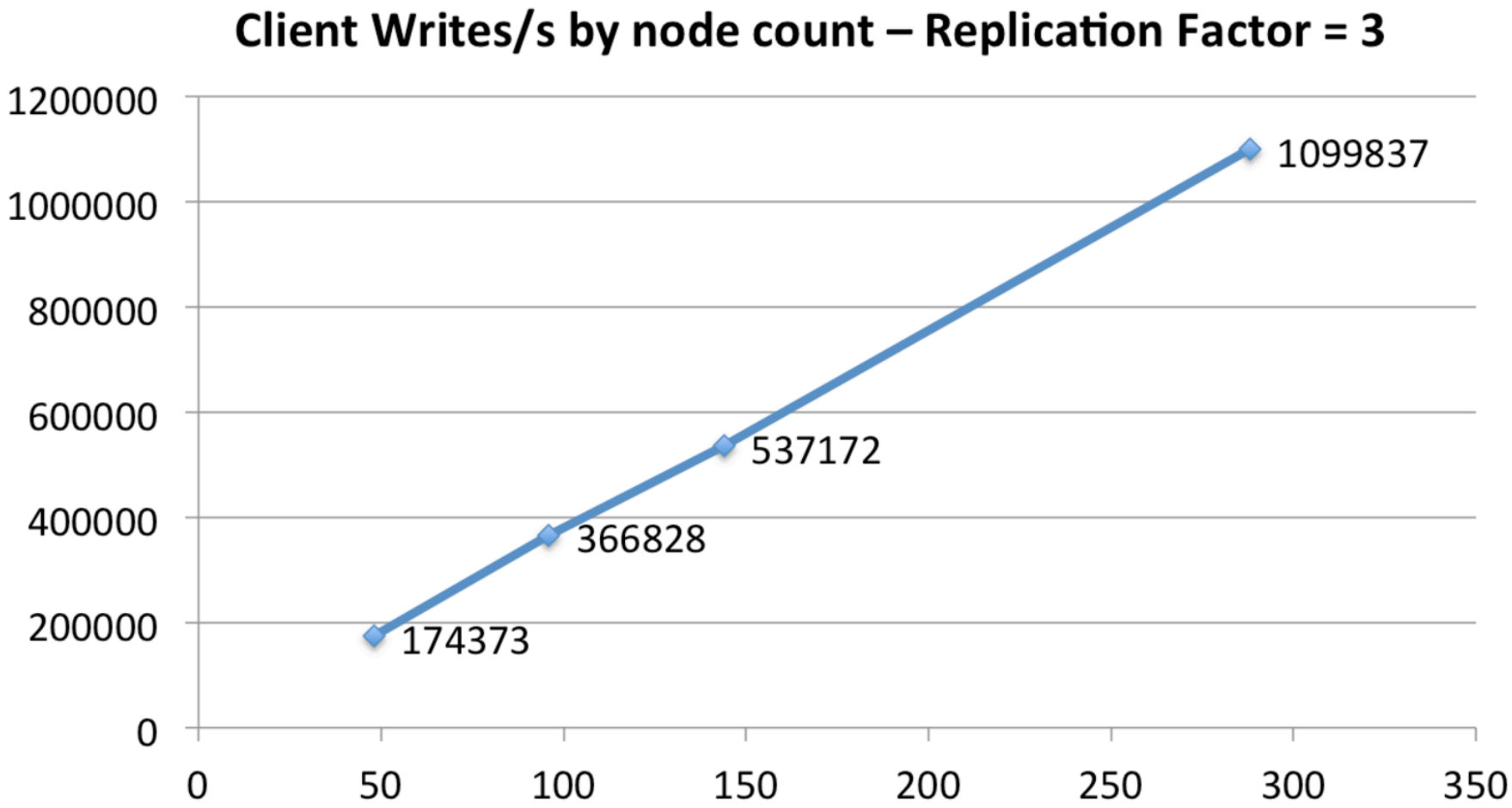
If a node or region goes offline, hinted handoff completes the write when the node comes back up. Nightly global compare and repair jobs ensure everything stays consistent.



Source: Netflix

Cassandra Scale Up

In Amazon EC2



Source: Netflix

The numbers

Per Node	48 Nodes	96 Nodes	144 Nodes	288 Nodes
Per Server Writes/s	10,900 w/s	11,460 w/s	11,900 w/s	11,456 w/s
Mean Server Latency	0.0117 ms	0.0134 ms	0.0148 ms	0.0139 ms
Mean CPU %Busy	74.4 %	75.4 %	72.5 %	81.5 %
Disk Read	5,600 KB/s	4,590 KB/s	4,060 KB/s	4,280 KB/s
Disk Write	12,800 KB/s	11,590 KB/s	10,380 KB/s	10,080 KB/s
Network Read	22,460 KB/s	23,610 KB/s	21,390 KB/s	23,640 KB/s
Network Write	18,600 KB/s	19,600 KB/s	17,810 KB/s	19,770 KB/s

Source: Netflix

Cassandra Model

- **Keyspaces** are roughly equivalent to SQL Databases
  - Encapsulate replication strategies
- **Column Families** roughly equivalent to SQL tables
- Generally a different approach vs SQL
  - Writes are cheap
  - Indexes are expensive
  - Normalization is not the goal

Cassandra Model cont.

- Inserts are the same as updates
  - No read first
- Data can be marked with a Time to Live (TTL)
  - Automatically deleted
- Deletes are not instant
  - Deleted rows are marked with a tombstone
  - Eventually cleaned up
  - Can re-appear if you do not run node repair after a node failure

CQL

- A variant of SQL written specifically for Cassandra
  - The preferred model of access
  - Replaces the old “Thrift” API
- Attempts to have some compatibility with normal SQL
  - e.g. you can use either KEYSPACE or TABLE interchangeably

CQL examples

SELECT name, occupation FROM users WHERE userid IN (199, 200, 207);

However, some queries are not permitted:

```
SELECT firstname, lastname FROM users WHERE
    birth_year = 1981 AND country = 'FR';
```

Requires a large scan of the database and cannot give a predictable time response:

ALLOW FILTERING will make this run anyway

INSERT / UPDATE

```
INSERT INTO NerdMovies (movie, director, main_actor, year)
    VALUES ('Serenity', 'Joss Whedon', 'Nathan Fillion', 2005)
     USING TTL 86400;
```

- Every row can have a specified expiry time
- Inserts work even if the data is already there, unless you specify:

```
INSERT INTO NerdMovies (movie, director, main_actor, year)
    VALUES ('Serenity', 'Joss Whedon', 'Nathan Fillion', 2005)
     IF NOT EXISTS
     USING TTL 86400;
```

This can have unpredictable timing because it requires read-before-write

Non-SQL data types

- Sets
  - CREATE TABLE cycling.cyclist_career_teams (id UUID PRIMARY KEY, lastname text, teams **set**<text>);
- Lists
  - CREATE TABLE cycling.upcoming_calendar (year int, month int, events **list**<text>, PRIMARY KEY (year, month));
- Maps
  - CREATE TABLE cycling.cyclist_teams (id UUID PRIMARY KEY, lastname text, firstname text, teams **map**<int,text>);
- Tuples
  - CREATE TABLE cycling.popular (rank int PRIMARY KEY, cinfo **tuple**<text,text,int>);

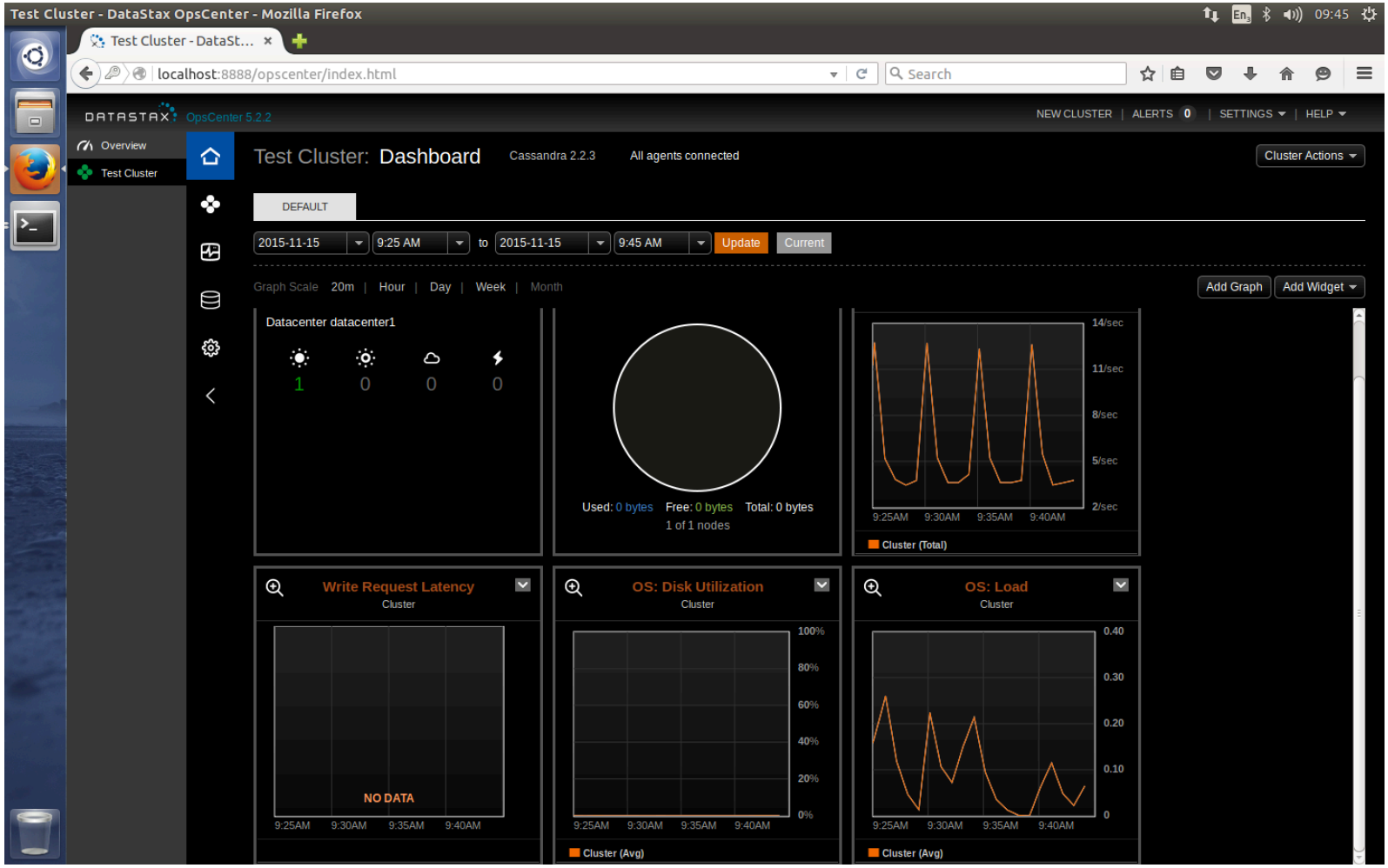
Direct support for JSON

```
INSERT INTO cycling.cyclist_category
JSON '{
  "category" : "GC",
  "points" : 780,
  "id" : "829aa84a-4bba-411f-
a4fb-38167a987cda",
  "lastname" : "SUTHERLAND" }';
```

cassandra.yaml

- Configuration of the major parts of the system
  - Datacentres, Racks, Cluster name
  - Authentication and Authorization
  - Partitioner
  - Data Storage location
  - Cacheing
  - Network topology and ports
  - Etc, etc

DataStax OpsCenter



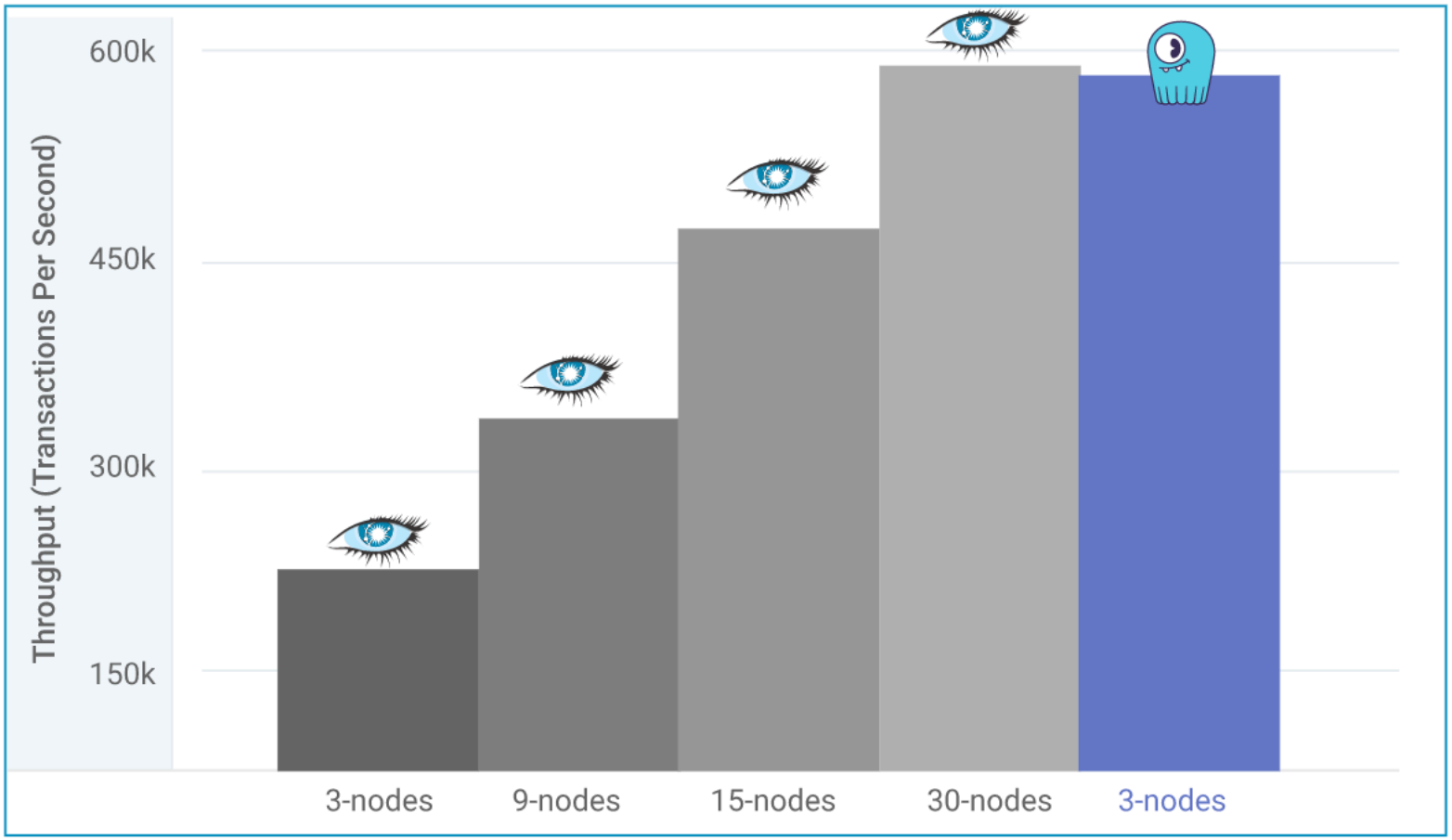
OpsCenter

- Part of DataStax Cassandra distribution
  - Community edition has limited features
  - Enterprise edition expands these
- Not open source, but free to use in the community edition
  - Requires an agent on each Cassandra node
  - It will install this via SSH if possible

ScyllaDB

- A C++ “clone” of Cassandra
- Also Open Source
- Claims to be significantly faster

Scylla vs Cassandra



Questions?