

# Cloud Computing and Big Data

## NoSQL databases

Oxford University  
Software Engineering  
Programme  
July 2019



# 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



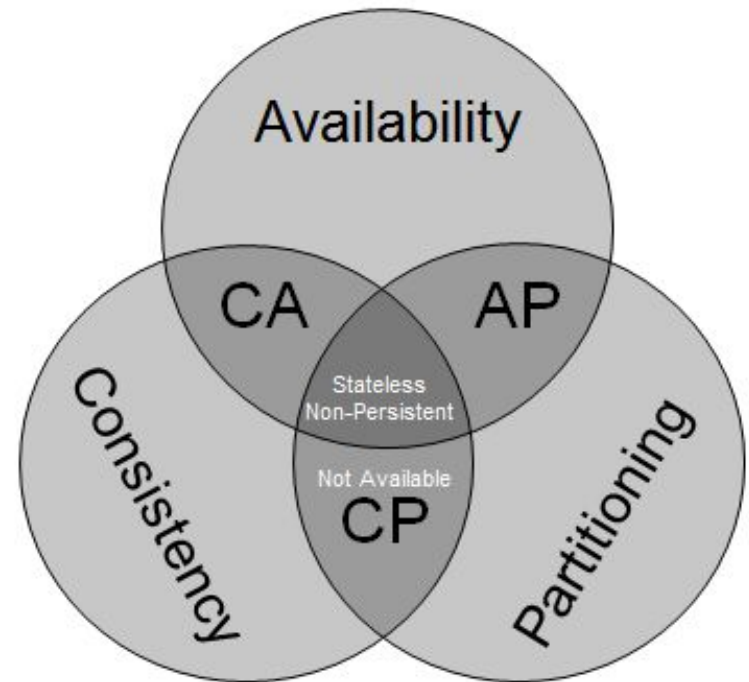
# NoSQL history

- 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

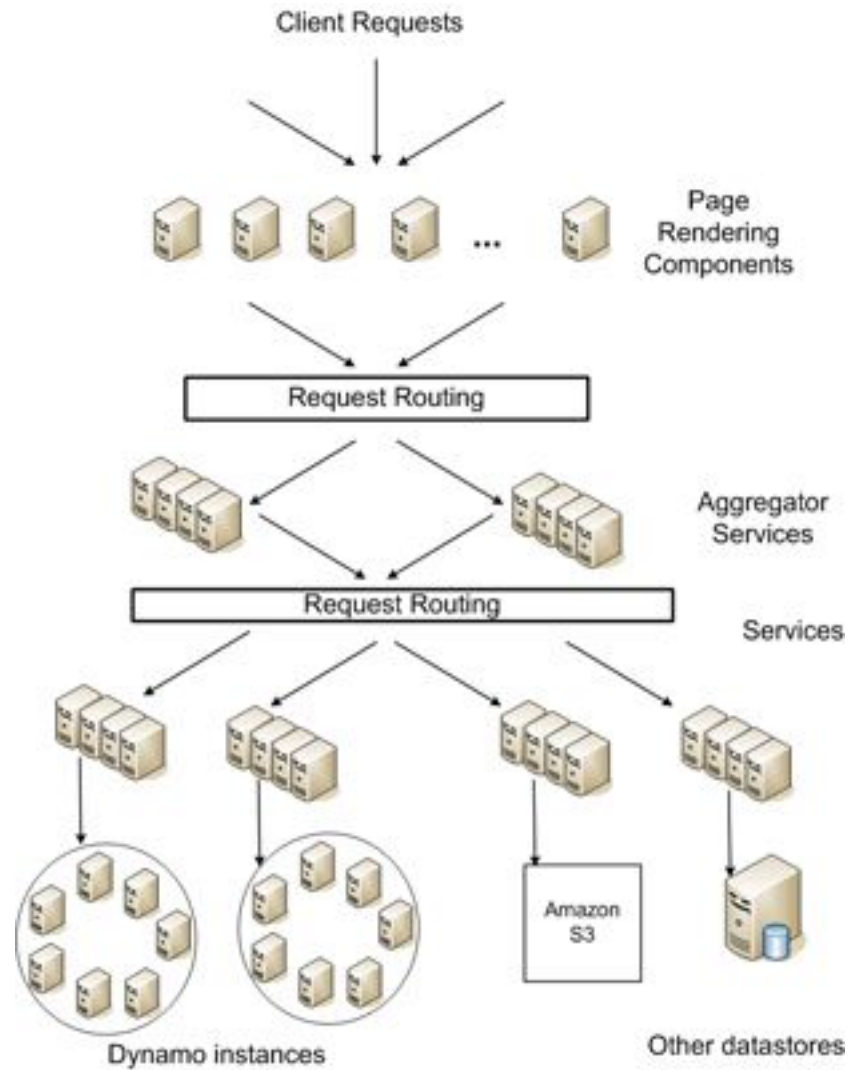


# NoSQL parents

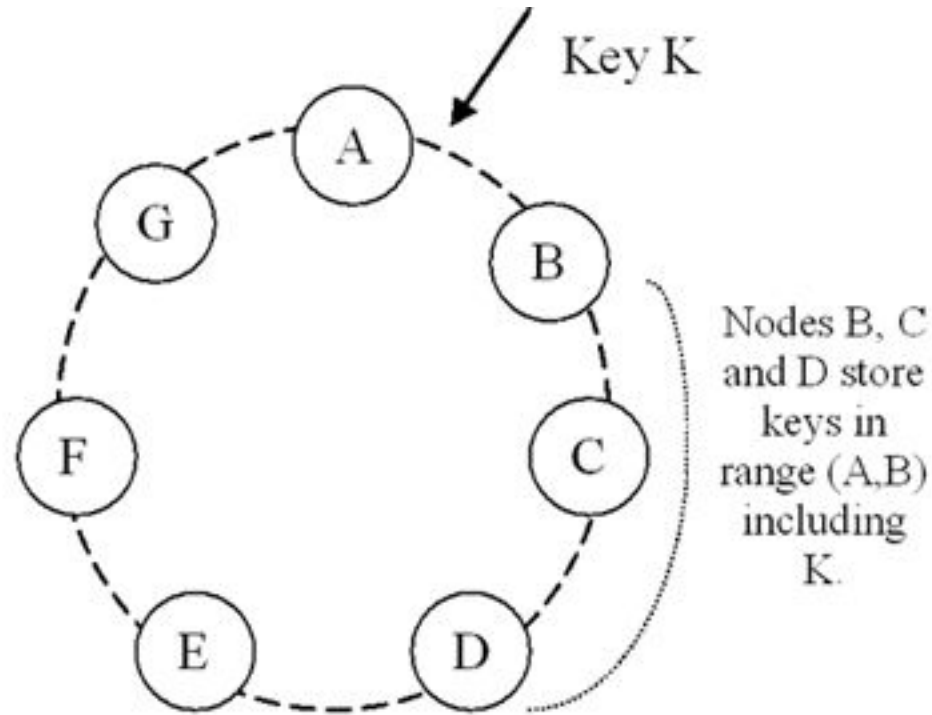
- Amazon Dynamo
  - Eventually consistent
- Google BigTable
  - Supporting very large rows
- LDM
  - Graph database



# Dynamo

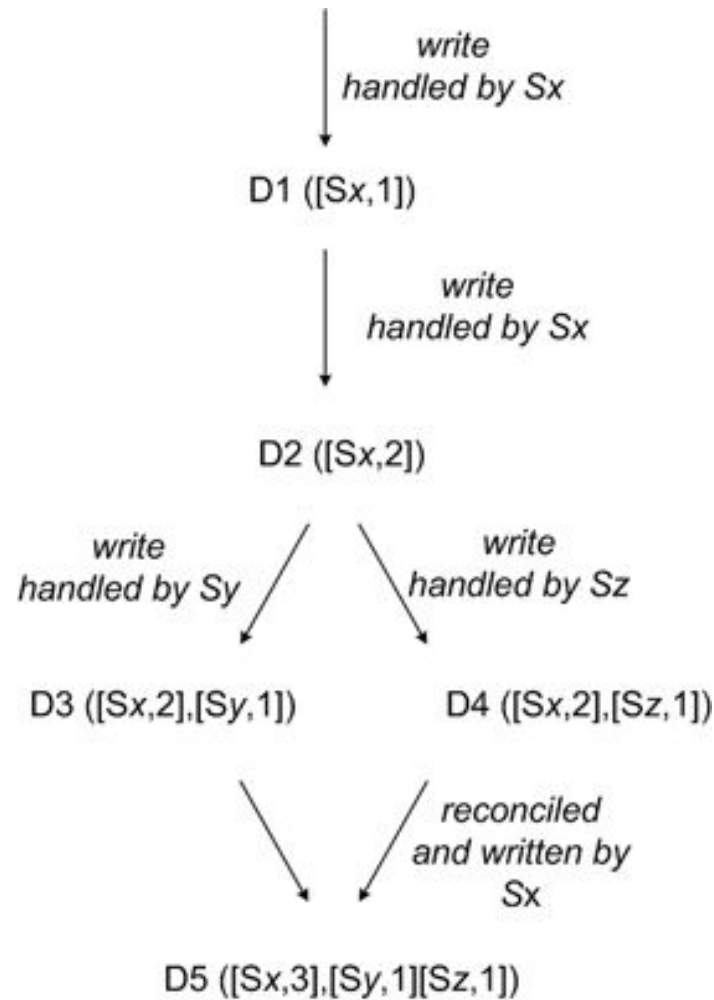


# Dynamo Model





# Reconciliation / Eventual Consistency



# 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?
<i>Crawl</i>	800	11%	1000	16	8	0%	No
<i>Crawl</i>	50	33%	200	2	2	0%	No
<i>Google Analytics</i>	20	29%	10	1	1	0%	Yes
<i>Google Analytics</i>	200	14%	80	1	1	0%	Yes
<i>Google Base</i>	2	31%	10	29	3	15%	Yes
<i>Google Earth</i>	0.5	64%	8	7	2	33%	Yes
<i>Google Earth</i>	70	–	9	8	3	0%	No
<i>Orkut</i>	9	–	0.9	8	5	1%	Yes
<i>Personalized Search</i>	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”

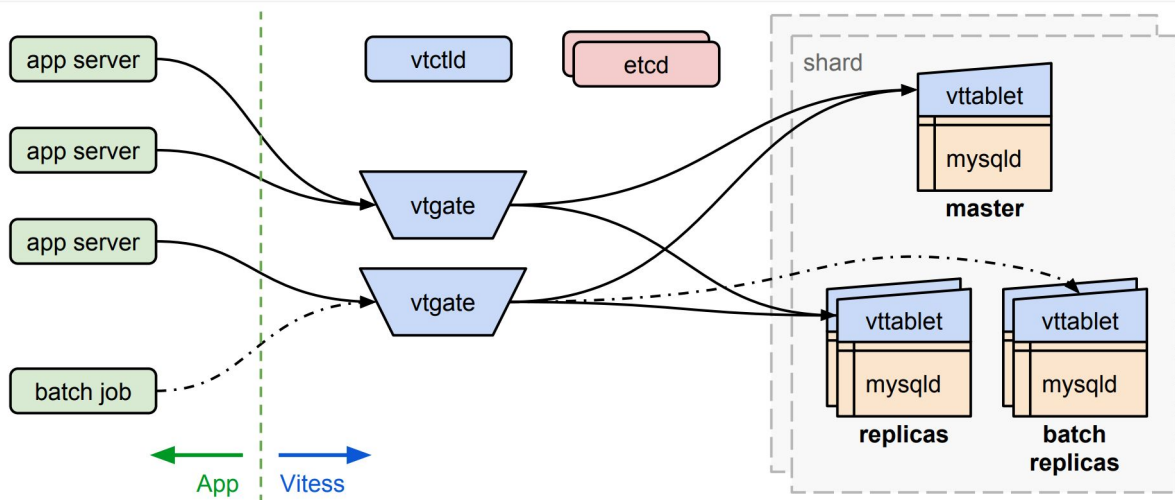
- ACID databases that aim to provide HA and Partition safety
  - VoltDB
  - NuoDB
  - Google Spanner
  - MemSQL
  - SAP HANA
- Also there are some backend engines for MySQL that aim to provide this:
  - MySQL Cluster
  - TokuDB



# Vitess

<https://github.com/youtube/vitess>

## Components



<https://freo.me/vitess-pres>

**Old Shards Running**

**New Shards Running**

**DB Read Availability**

**DB Write Availability**

**DB Write Downtime**

**DB Write Availability**

**< 5 seconds**

# 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



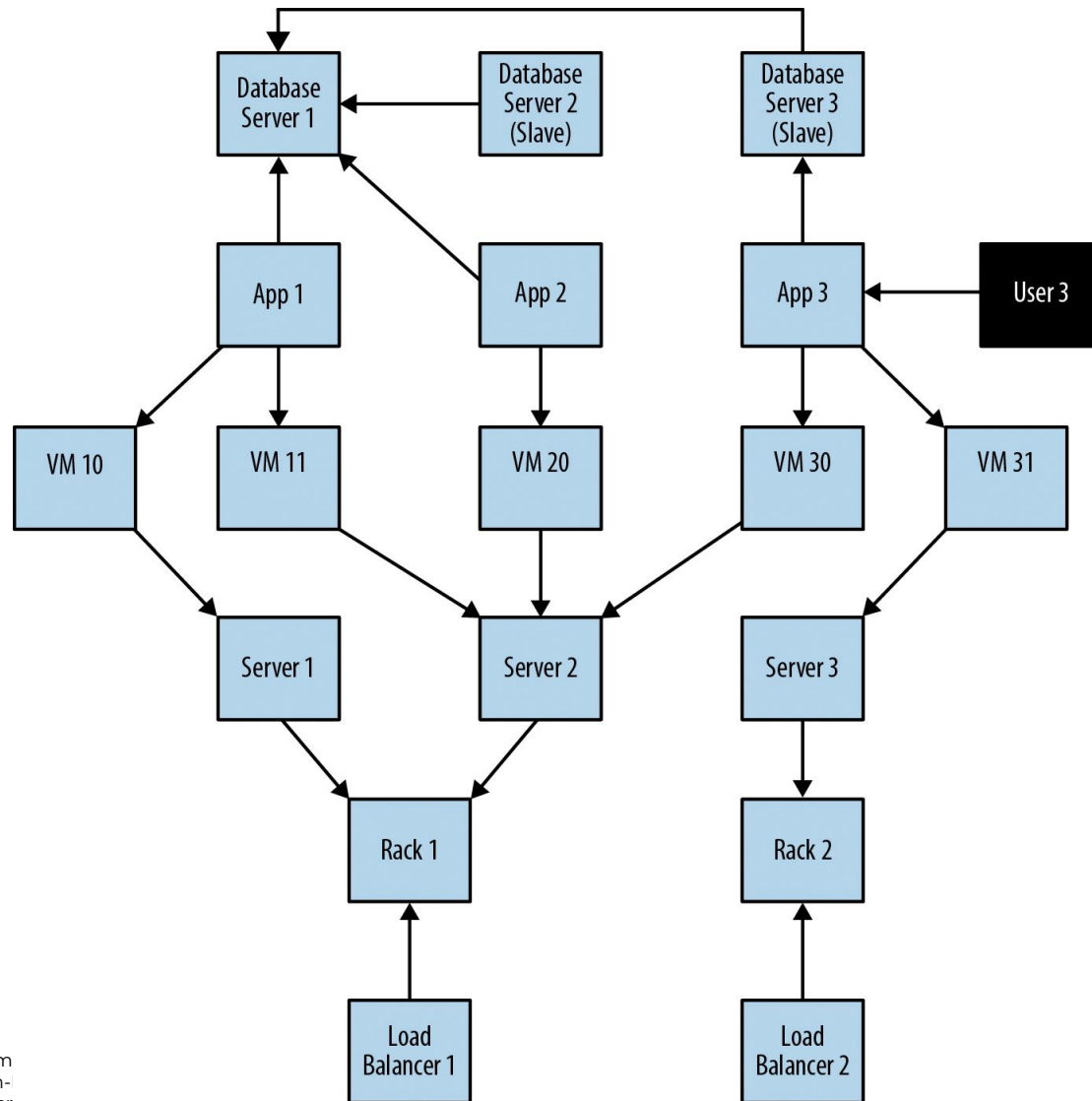
# Key Value databases

- 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

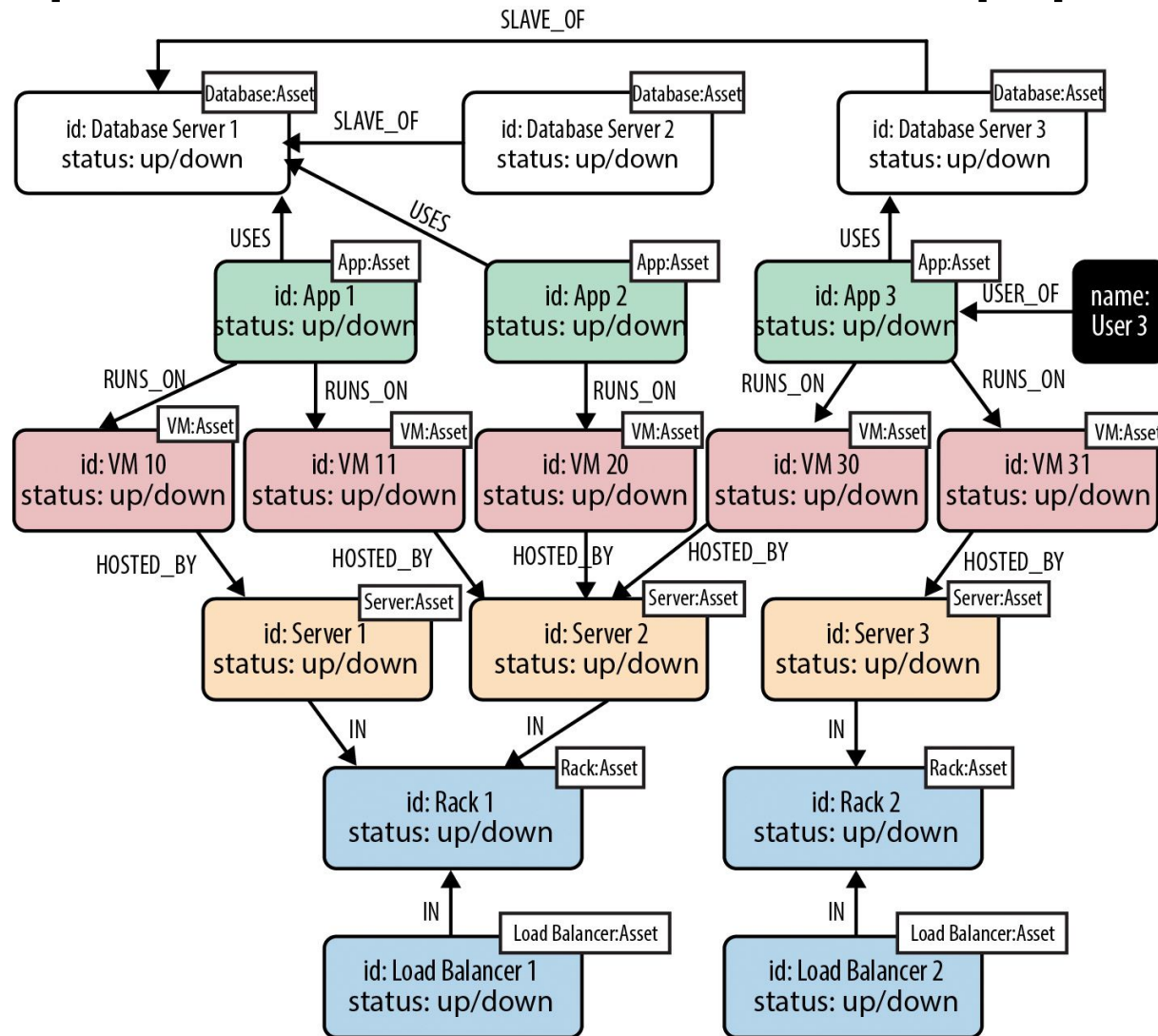




# Graph Databases



# Graph Database mapping



# Do We Need Specialized Graph Databases? Benchmarking Real-Time Social Networking Applications

Anil Pacaci, Alice Zhou, Jimmy Lin, and M. Tamer Özsu

David R. Cheriton School of Computer Science

University of Waterloo

Waterloo, Ontario, Canada

{apacaci,r32zhou,jimmylin,tamer.ozsu}@uwaterloo.ca

## ABSTRACT

With the advent of online social networks, there is an increasing demand for storage and processing of graph-structured data. Social networking applications pose new challenges to data management systems due to demand for real-time querying and manipulation of the graph structure. Recently, several systems specialized systems for graph-structured data have been introduced. However, whether we should abandon mature RDBMS technology for graph databases remains an ongoing discussion. In this paper we present an graph database benchmarking architecture built on the existing LDBC Social Network Benchmark. Our proposed architecture stresses the systems with an interactive transactional workload to better simulate the real-time nature of social networking applications. Using this improved architecture, we evaluated a selection of specialized graph databases, RDF stores, and RDBMSes adapted for graphs. We do not find that specialized graph databases provide definitively better performance.

in robust RDBMS technology, and (ii) its dominance in data analytic ecosystems in enterprise settings. Studies show that special-purpose graph analytics engines do not necessarily provide the best performance across all scenarios. Indeed, relational models can provide competitive performance for various graph analytic tasks, especially on single node, out-of-memory settings [5, 8].

Similar arguments can be made for OLTP-like graph workloads; however, there are no comprehensive studies of existing systems for real-world, dynamic graph workloads such as online social networks. Many studies focus on comparisons between different graph database engines and graph analytics systems [7, 10]. Although there are some studies comparing graph databases with relational models [2, 3, 6, 11], the real-time aspect of graph applications is mostly ignored and more complex graph traversals are not tested.

Our objective in this paper is twofold: (i) to propose and implement an improved graph database benchmarking architecture for real-time transaction processing and (ii) to present an experimental comparison of various graph data management solutions in online



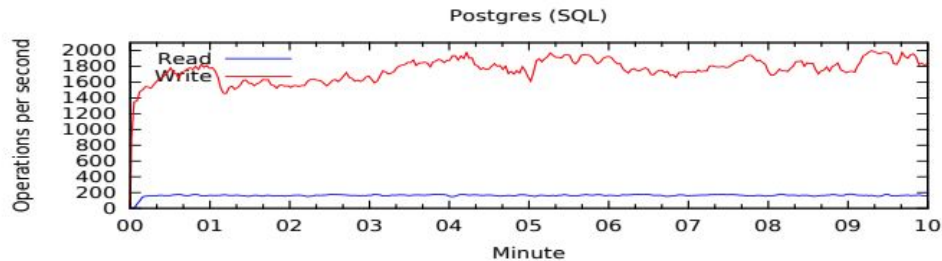
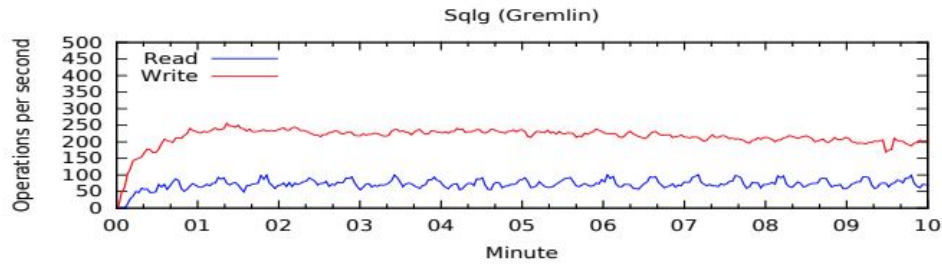
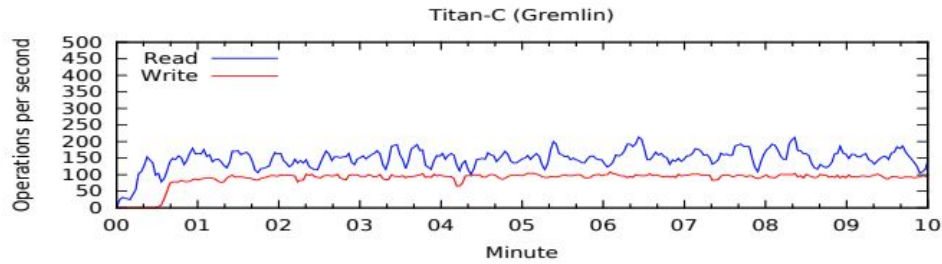
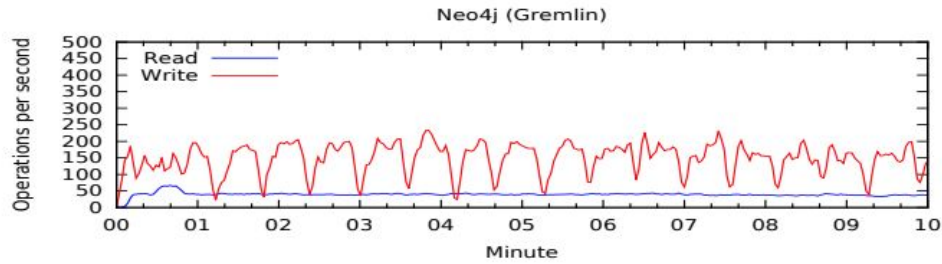
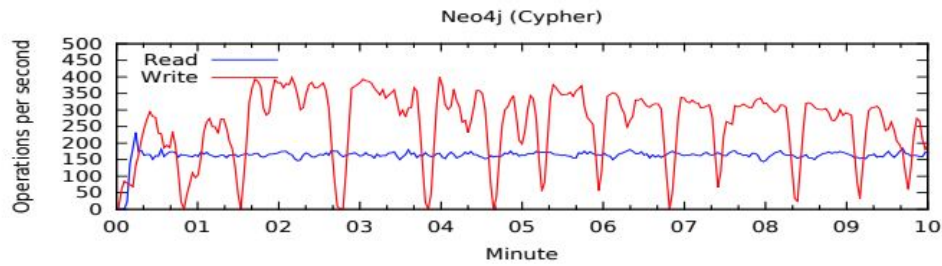


Table 2: Query Latencies in ms – Scale Factor 3

System	Neo4j		Titan-C	Titan-B	Sqlg	Postgres	Virtuoso	
Query Language	Cypher	Gremlin	Gremlin	Gremlin	Gremlin	SQL	SQL	SPARQL
Point lookup	9.08	122	39	65	16.1	0.25	0.35	3
1-hop	12.82	101	240	223	34	1.4	2.15	1.23
2-hop	368	275	439	1271	2526	29	11.55	16.62
Shortest Path	21	4813	10732	13948	10243	2242	4.81	26

Table 3: Query Latencies in ms – Scale Factor 10

System	Neo4j		Titan-C	Titan-B	Sqlg	Postgres	Virtuoso	
Query Language	Cypher	Gremlin	Gremlin	Gremlin	Gremlin	SQL	SQL	SPARQL
Point lookup	11.16	177	42	236	16.9	0.32	0.41	3
1-hop	14.1	377	129	2117	43	1.62	2.22	1.71
2-hop	579	683	1570	12978	4408	46	15.92	52
Shortest Path	16	4053	17379	-	7003	3648	7.09	32



# Top ten databases

283 systems in ranking, November 2015

Rank			DBMS	Database Model	Score		
Nov 2015	Oct 2015	Nov 2014			Nov 2015	Oct 2015	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>

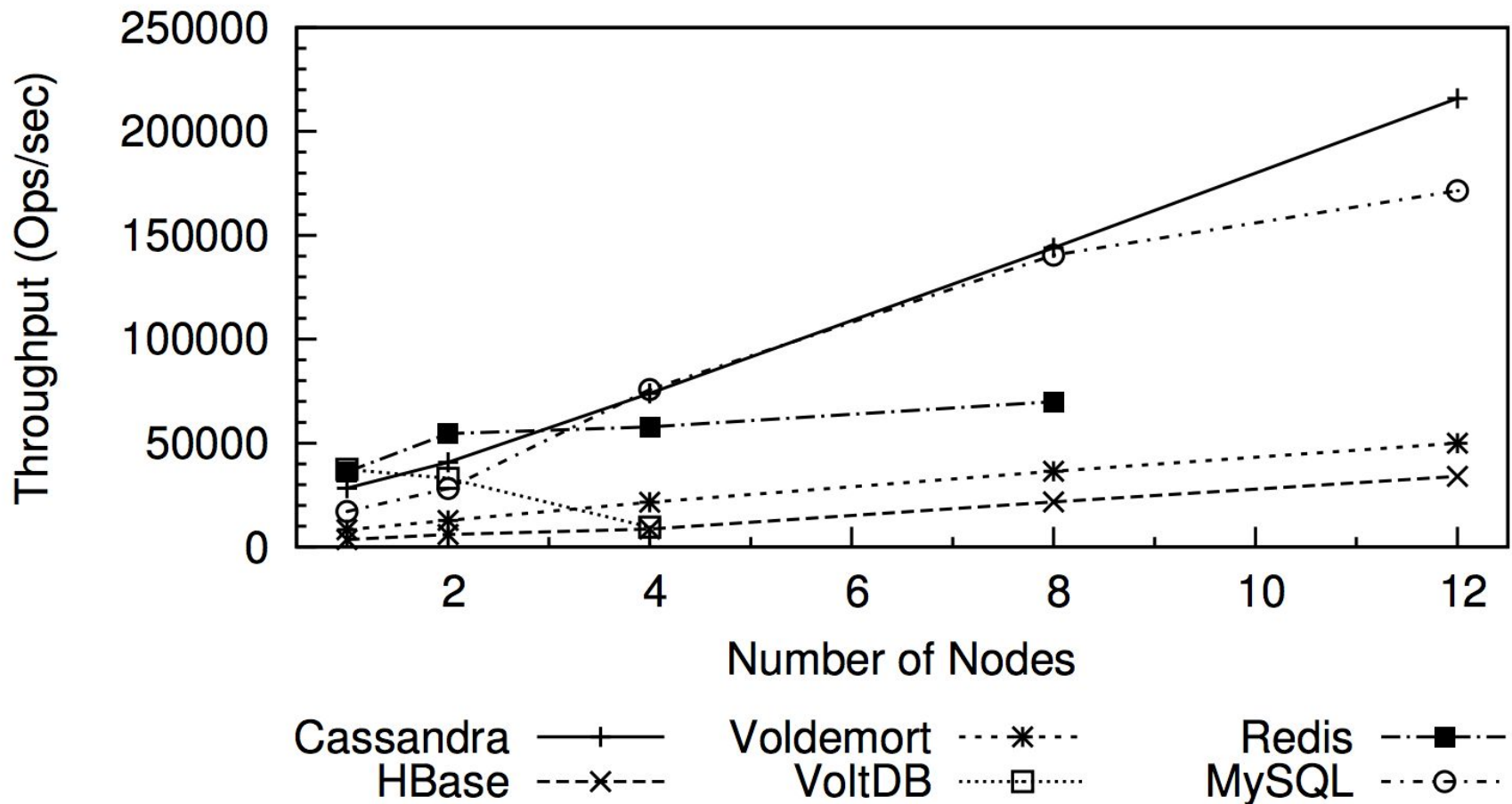


# Next 20

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.

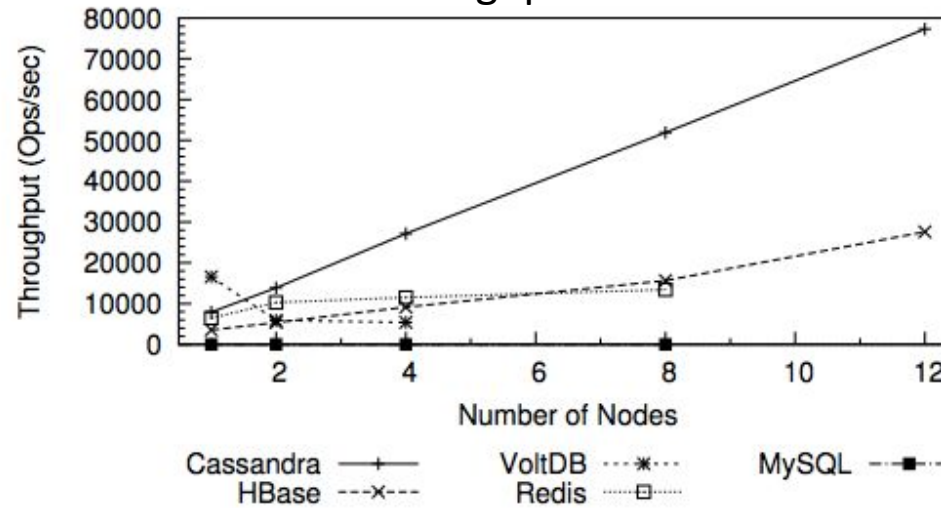


See <http://creativecommons.org/licenses/by-nc-sa/4.0/>

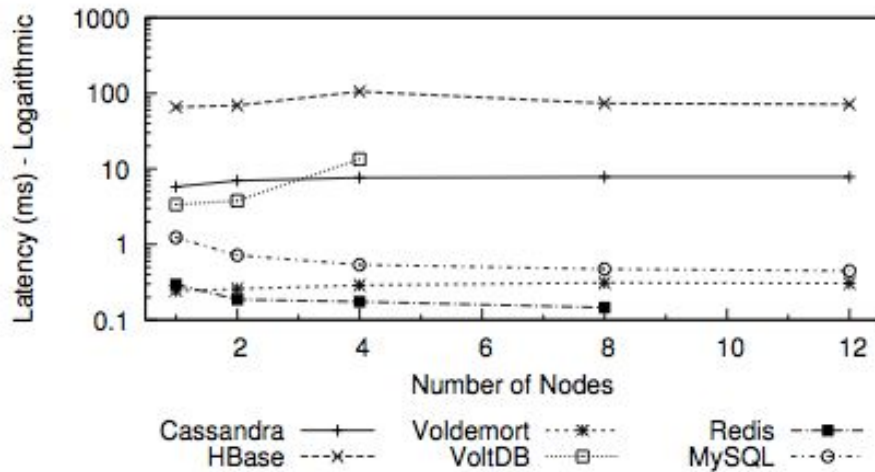
# More performance (2012)

Read/Scan/Write workload

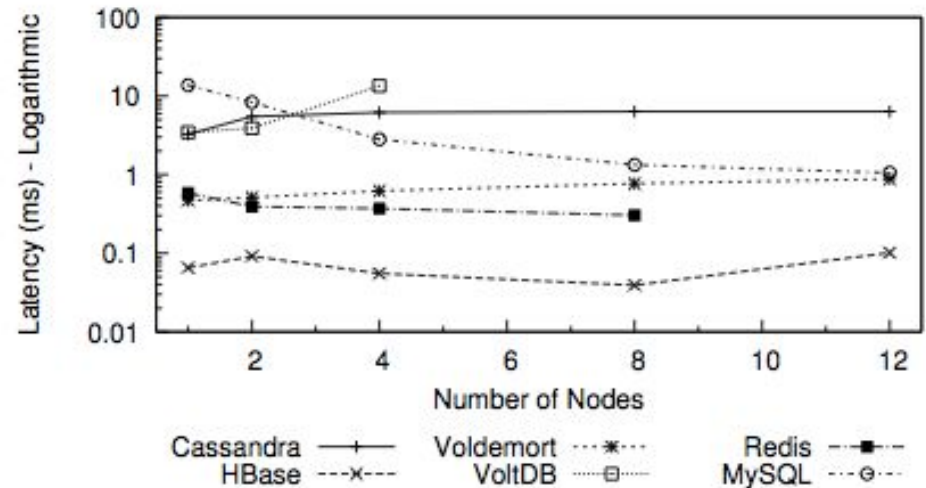
Throughput



Read Latency



Write Latency

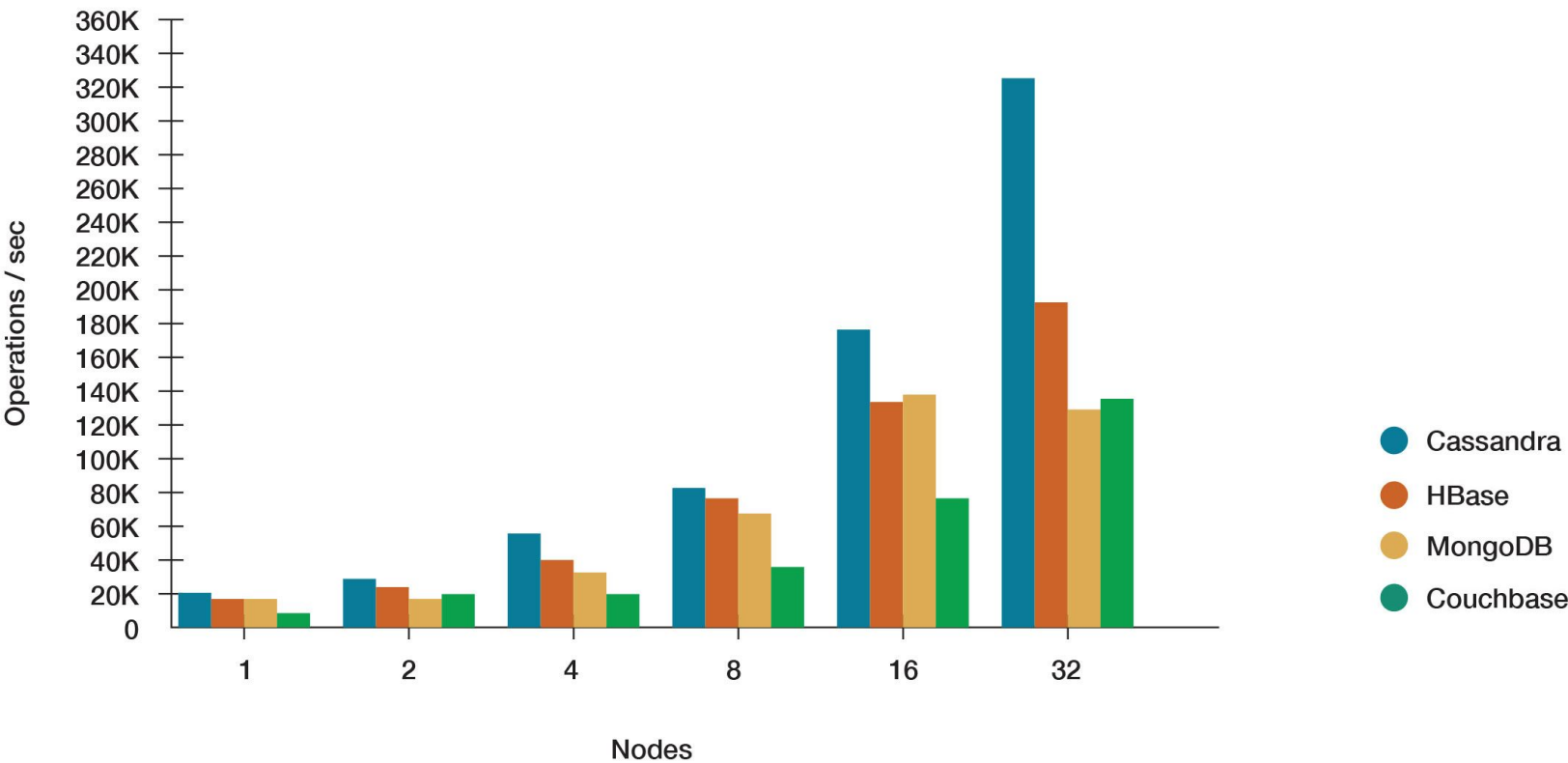




# Summary of Performance benchmark (2012)

- **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





Nodes	Cassandra	HBase	MongoDB	Couchbase
1	18,683.43	15,617.98	8,368.44	13,761.12
2	31,144.24	23,373.93	13,462.51	26,140.82
4	53,067.62	38,991.82	18,038.49	40,063.34
8	86,924.94	74,405.64	34,305.30	76,504.40
16	173,001.20	143,553.41	73,335.62	131,887.99
32	326,427.07	296,857.36	134,968.87	192,204.94

# Questions?



© Paul Fremantle 2015. This work is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License  
See <http://creativecommons.org/licenses/by-nc-sa/4.0/>