Cloud Computing and Big Data

NoSQL databases

Oxford University
Software Engineering Programme
July 2020

Contents

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

Why NoSQL?

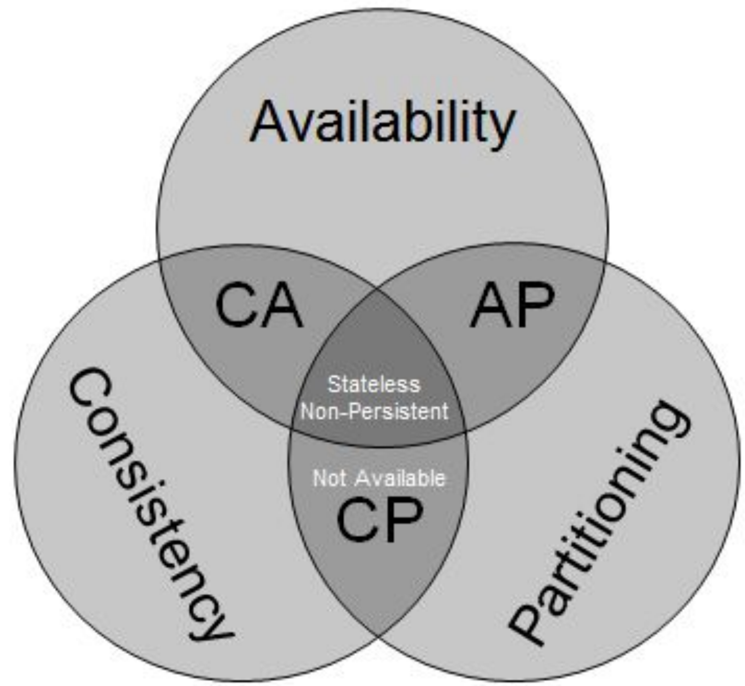
- Availability
  - Need better scaling capabilities
  - Elasticity
- Different schema approaches
  - Graphs, Key Values, Document, Sparse Columns, etc
- More appropriate balance in read/write performance
- Better integration with REST/SOA/Cloud

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

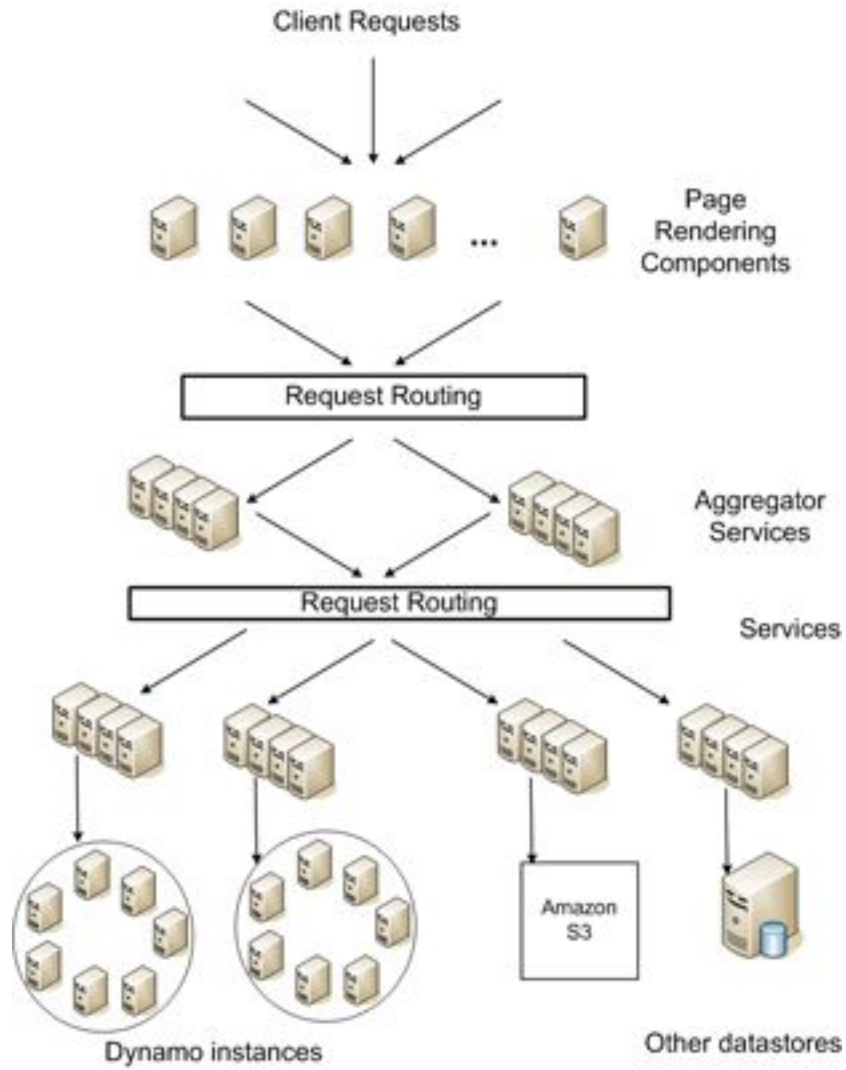


http://blog.mccrory.me/2010/11/03/cap-theorem-and-the-clouds/

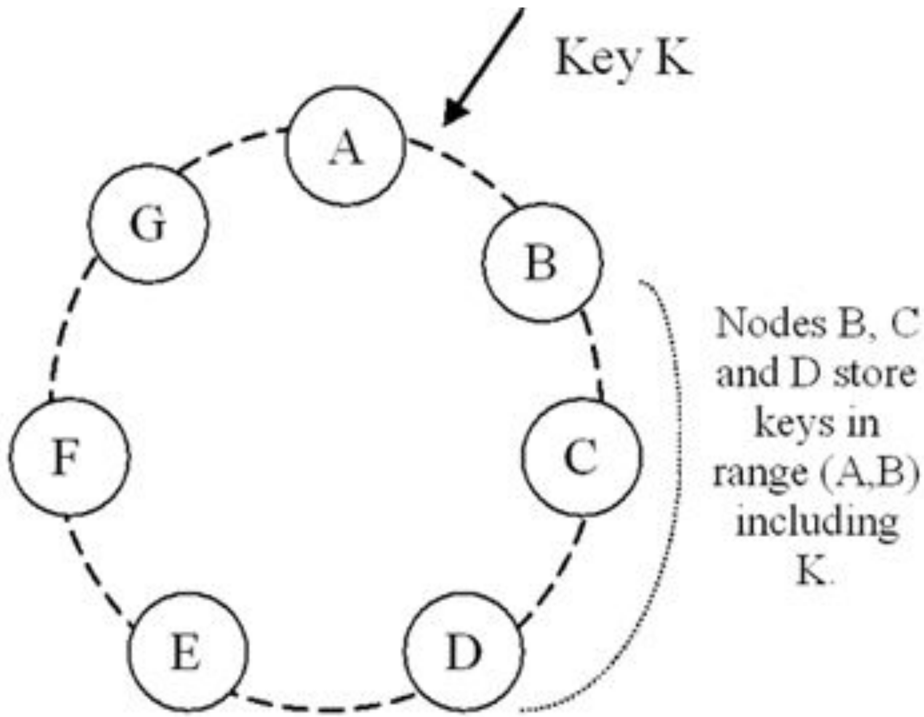
NoSQL parents

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

Dynamo

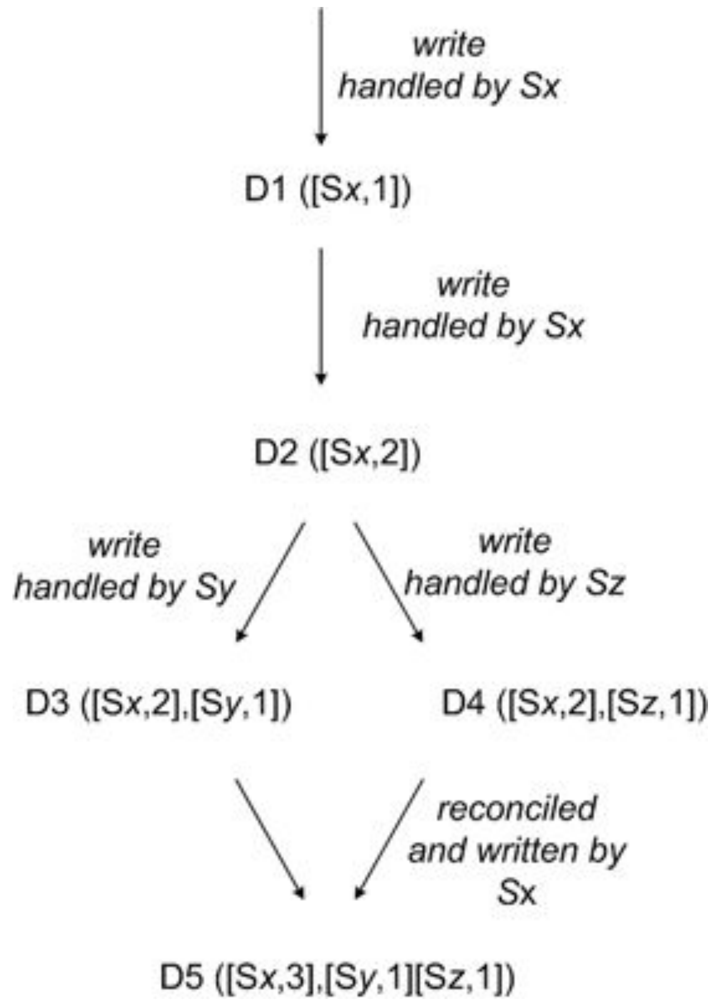

Client Requests
...
Page
Rendering
Components
Request Routing
Aggregator
Services
Request Routing
Services
Amazon
S3
Dynamo instances
Other datastores

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

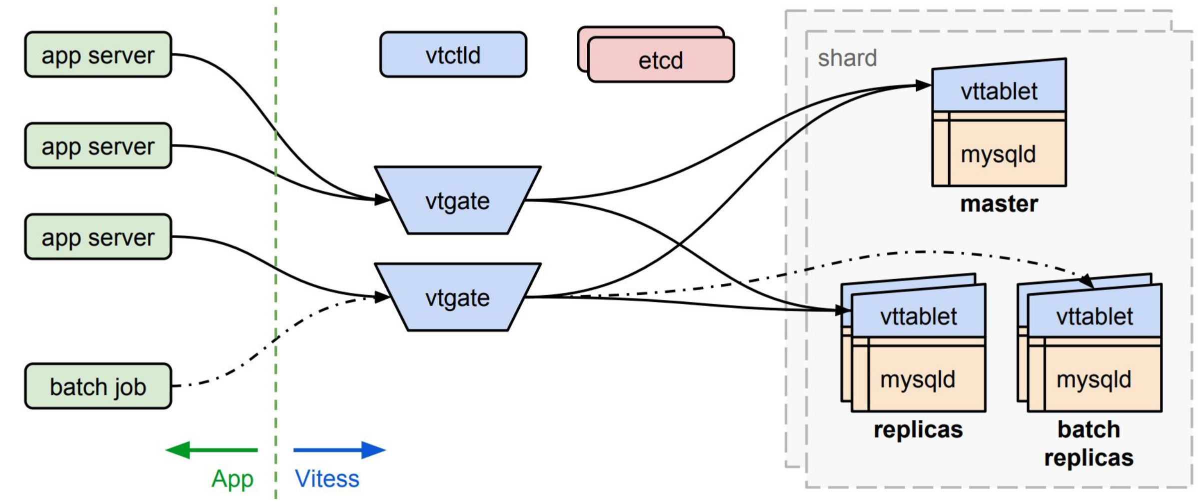
“NewSQL”

- 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

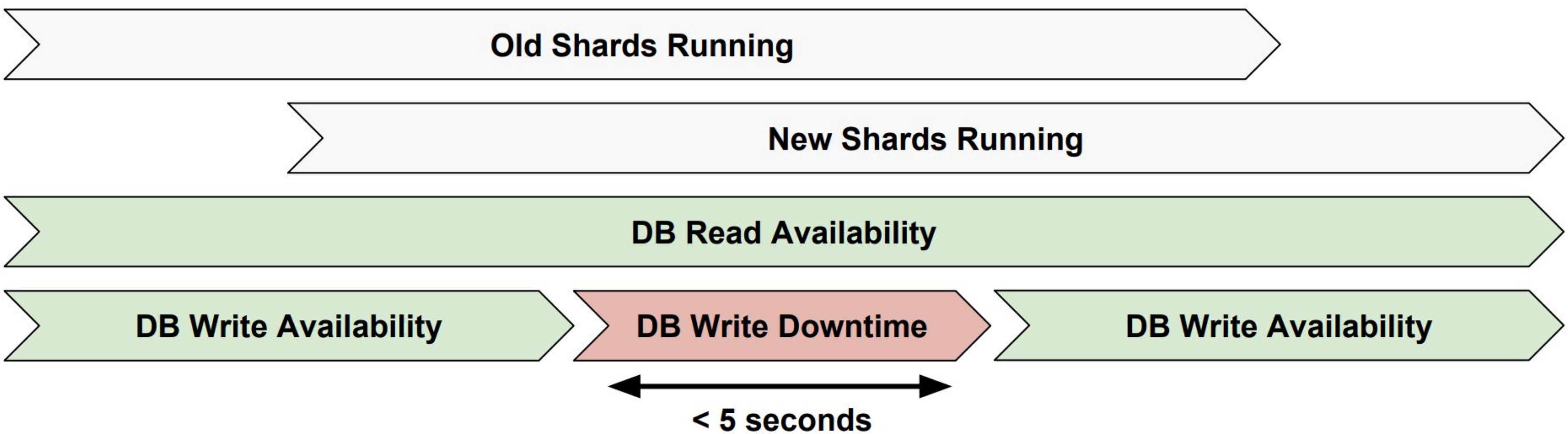
Vitess

https://github.com/youtube/vitess

Components



https://freo.me/vitess-pres



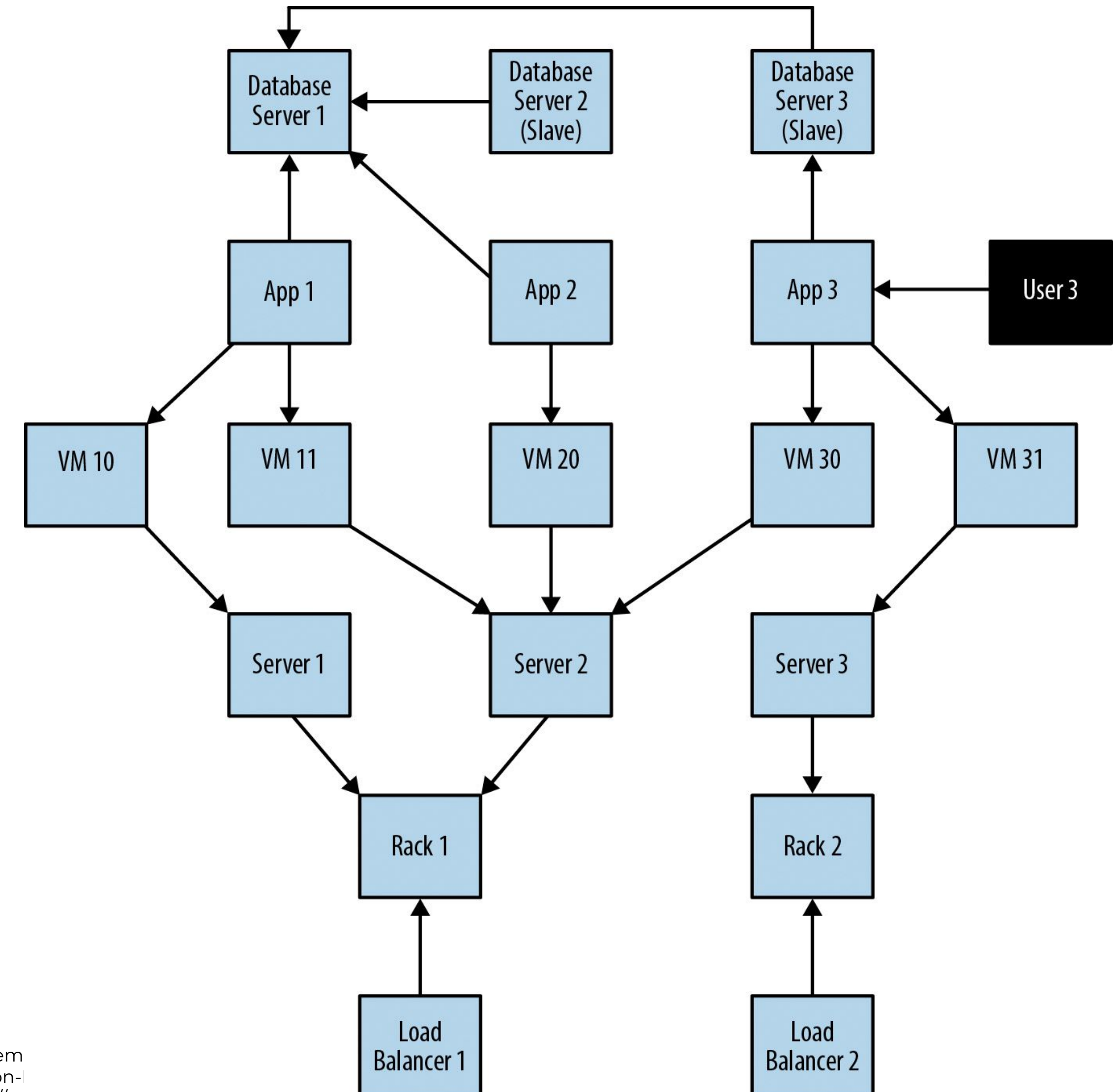
In Memory Databases

- 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

Key Value databases

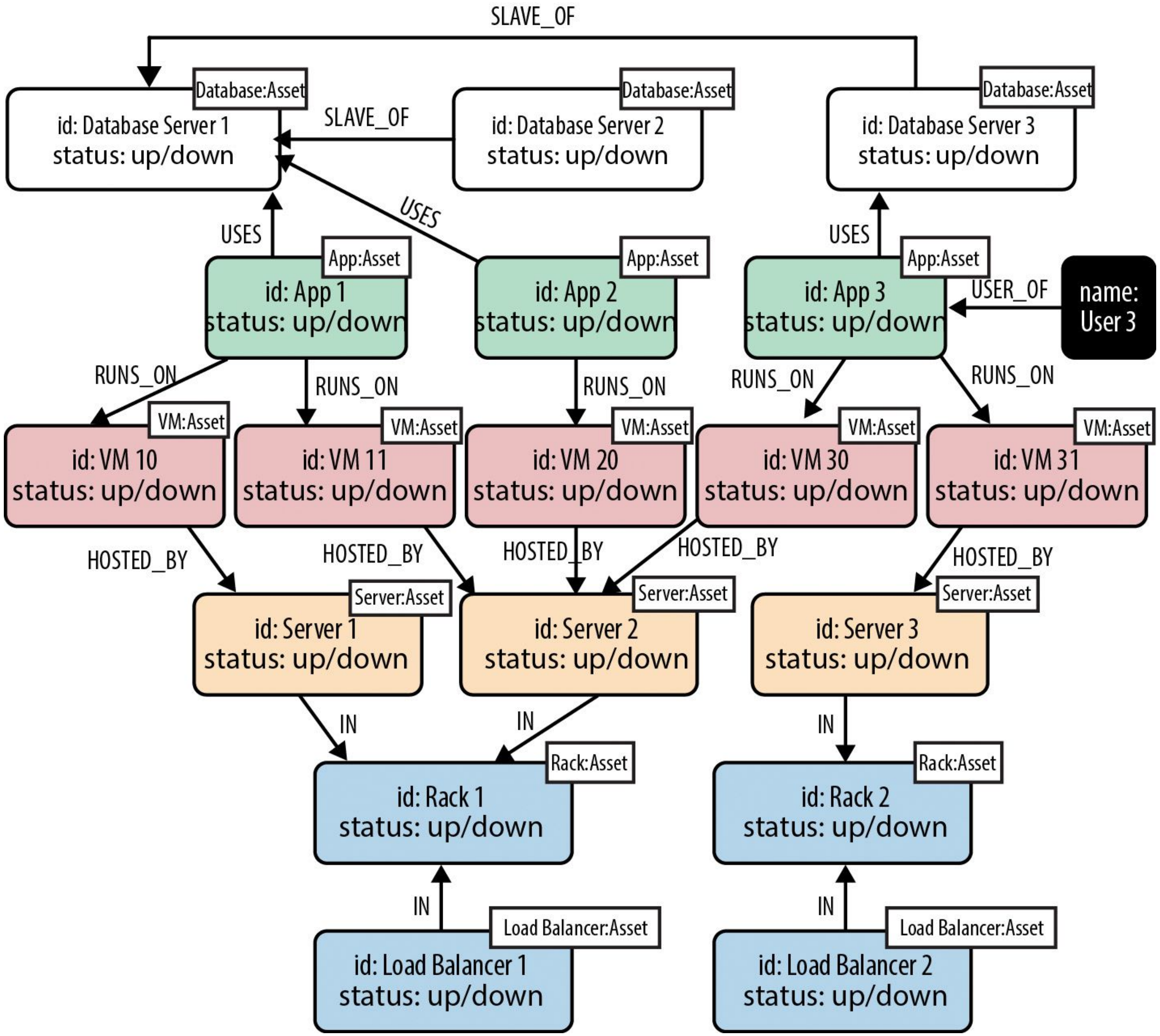
- A persistent associative array or dictionary
- Simple access and fits well with programming models (especially MR)
- Indexing on other data is not often possible and can be slow

Graph Databases



Source: neo4j

Graph Database mapping



Source: neo4j

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

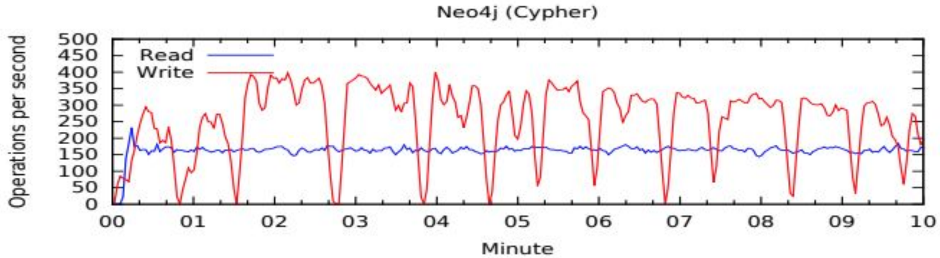
ABSTRACT

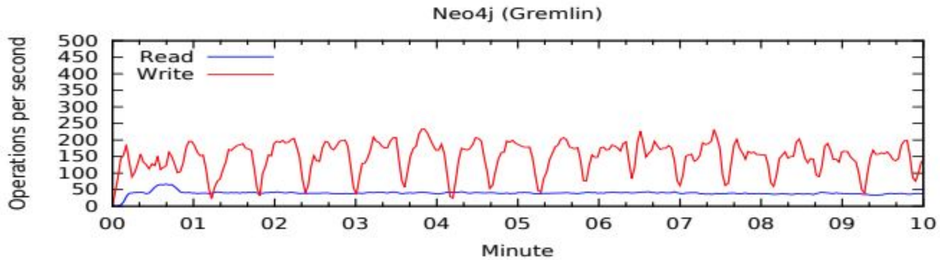
With the advent of online social networks, there is an increasing demand for storage and processing of graph-structured data. Social networking applications pose new challenges to data management systems due to demand for real-time querying and manipulation of the graph structure. Recently, several systems 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.

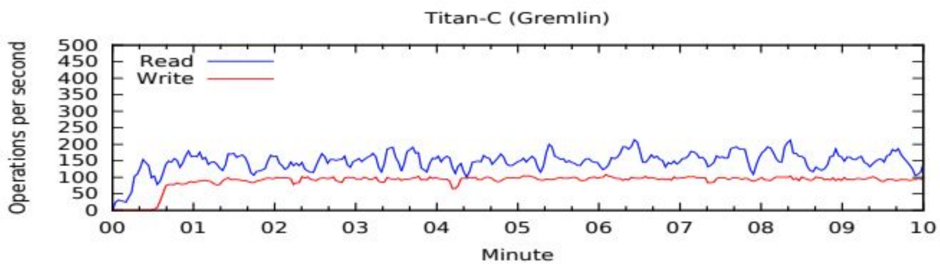
in robust RDBMS technology, and (ii) its dominance in data analytic ecosystems in enterprise settings. Studies show that special-purpose graph analytics engines do not necessarily provide the best performance across all scenarios. Indeed, relational models can provide competitive performance for various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

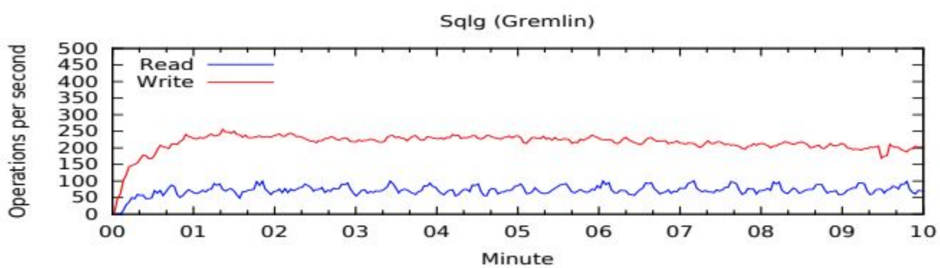
Similar arguments can be made for OLTP-like graph workloads; however, there are no comprehensive studies of existing systems for real-world, dynamic graph workloads such as online social networks. Many studies focus on comparisons between different graph database engines and graph analytics systems [7, 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









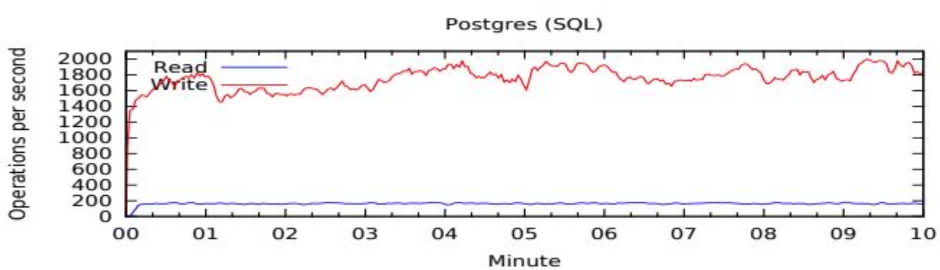


Table 2: Query Latencies in ms — Scale Factor 3

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

356 systems in ranking, June 2020

Rank			DBMS	Database Model	Score		
Jun 2020	May 2020	Jun 2019			Jun 2020	May 2020	Jun 2019
1.	1.	1.	Oracle	Relational, Multi-model	1343.59	-1.85	+44.37
2.	2.	2.	MySQL	Relational, Multi-model	1277.89	-4.75	+54.26
3.	3.	3.	Microsoft SQL Server	Relational, Multi-model	1067.31	-10.99	-20.45
4.	4.	4.	PostgreSQL	Relational, Multi-model	522.99	+8.19	+46.36
5.	5.	5.	MongoDB	Document, Multi-model	437.08	-1.92	+33.17
6.	6.	6.	IBM Db2	Relational, Multi-model	161.81	-0.83	-10.39
7.	7.	7.	Elasticsearch	Search engine, Multi-model	149.69	+0.56	+0.86
8.	8.	8.	Redis	Key-value, Multi-model	145.64	+2.17	-0.48
9.	9.	↑ 11.	SQLite	Relational	124.82	+1.78	-0.07
10.	↑ 11.	10.	Cassandra	Wide column	119.01	-0.15	-6.17

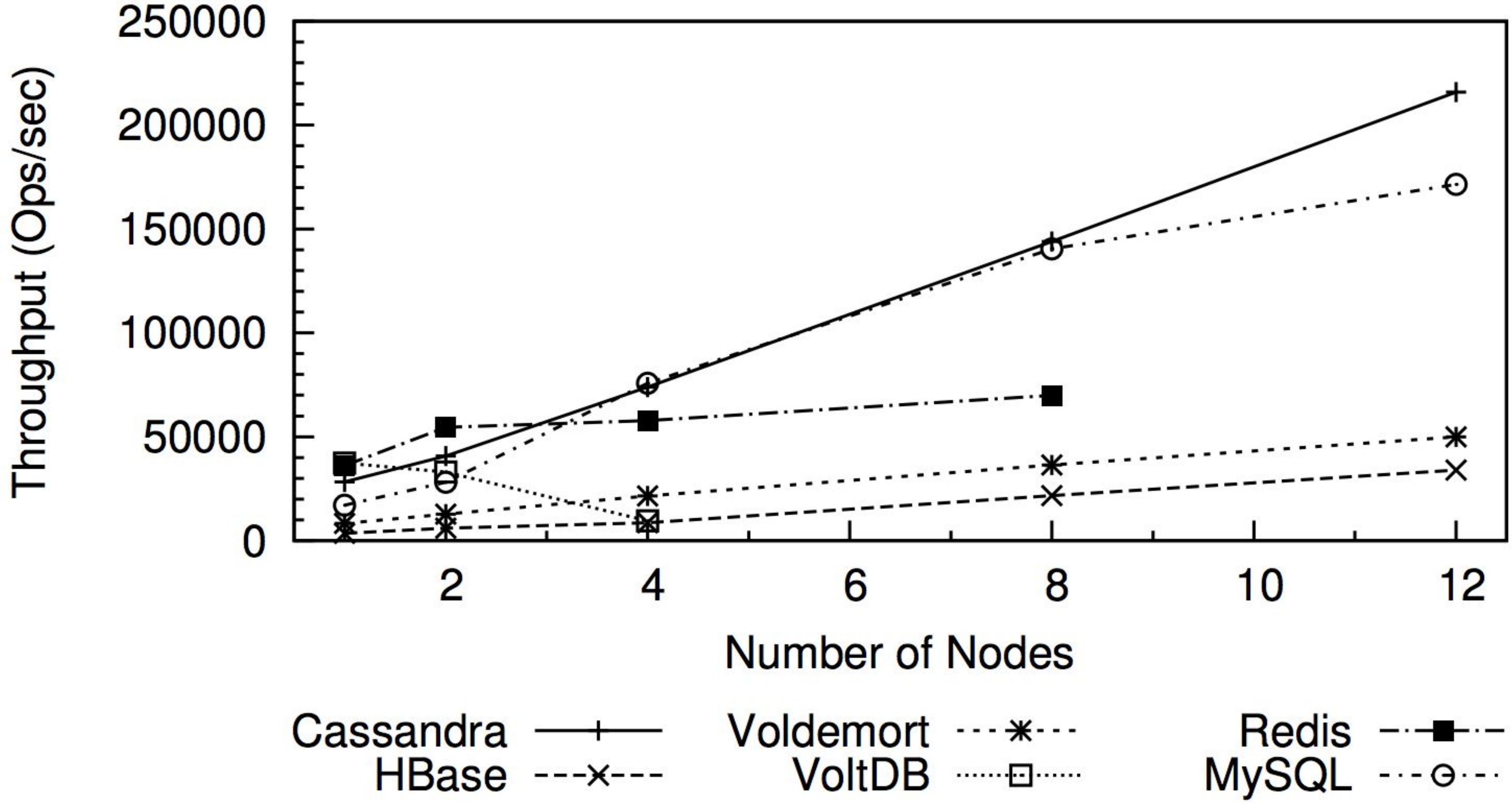
http://db-engines.com/en/ranking

Next 20

11.	10.	9.	Microsoft Access	Relational	117.18	-2.72	-23.83
12.	12.	12.	MariaDB	Relational, Multi-model	89.79	-0.30	+4.59
13.	13.	13.	Splunk	Search engine	88.08	+0.33	+3.46
14.	14.	14.	Hive	Relational	78.65	-2.89	-0.40
15.	15.	15.	Teradata	Relational, Multi-model	73.28	-0.60	-3.36
16.	16.	20.	Amazon DynamoDB	Multi-model	64.87	+0.15	+9.61
17.	17.	21.	SAP Adaptive Server	Relational	53.09	-0.90	-2.03
18.	18.	16.	Solr	Search engine	51.26	-1.32	-9.22
19.	20.	19.	SAP HANA	Relational, Multi-model	50.82	+0.29	-5.56
20.	19.	18.	FileMaker	Relational	50.16	-0.80	-7.64
21.	22.	17.	HBase	Wide column	48.73	-0.99	-9.30
22.	21.	22.	Neo4j	Graph	48.27	-1.49	-1.28
23.	23.	24.	Microsoft Azure SQL Database	Relational, Multi-model	47.78	+5.03	+18.77
24.	24.	25.	Microsoft Azure Cosmos DB	Multi-model	30.80	+0.13	+2.56
25.	25.	23.	Couchbase	Document, Multi-model	29.14	+0.56	-4.22
26.	26.	28.	Google BigQuery	Relational	28.29	+0.70	+5.16
27.	27.	26.	Memcached	Key-value	24.81	+0.88	-3.20
28.	28.	27.	Informix	Relational, Multi-model	24.38	+0.54	-2.44
29.	31.	31.	Amazon Redshift	Relational	21.24	+0.97	+0.97
30.	29.	34.	InfluxDB	Time Series	21.18	+0.26	+3.19

Performance (2012)

50%/50% reads/writes

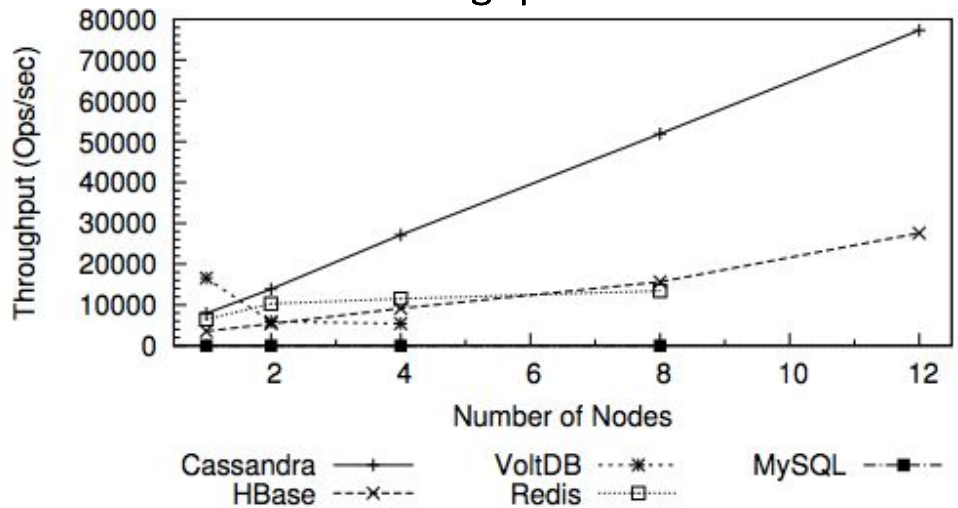


Rabl, Tilmann, et al. "Solving big data challenges for enterprise application performance management." Proceedings of the VLDB Endowment 5.12 (2012): 1724-1735.

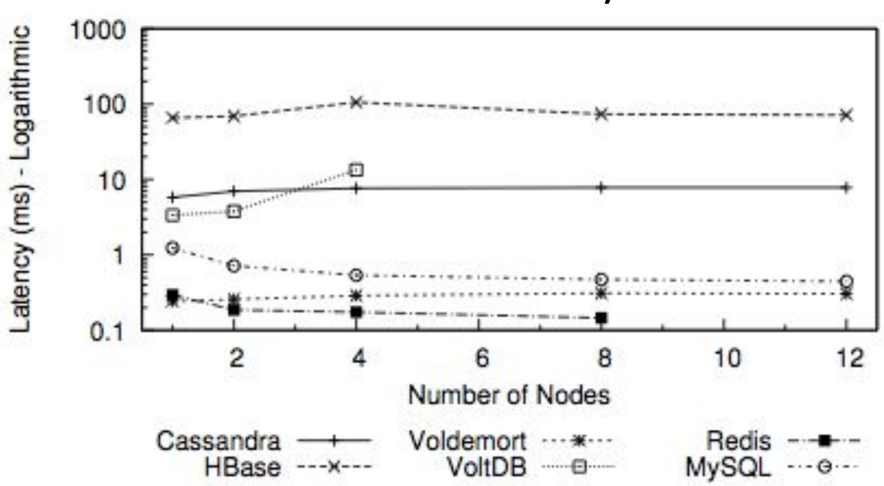
More performance (2012)

Read/Scan/Write workload

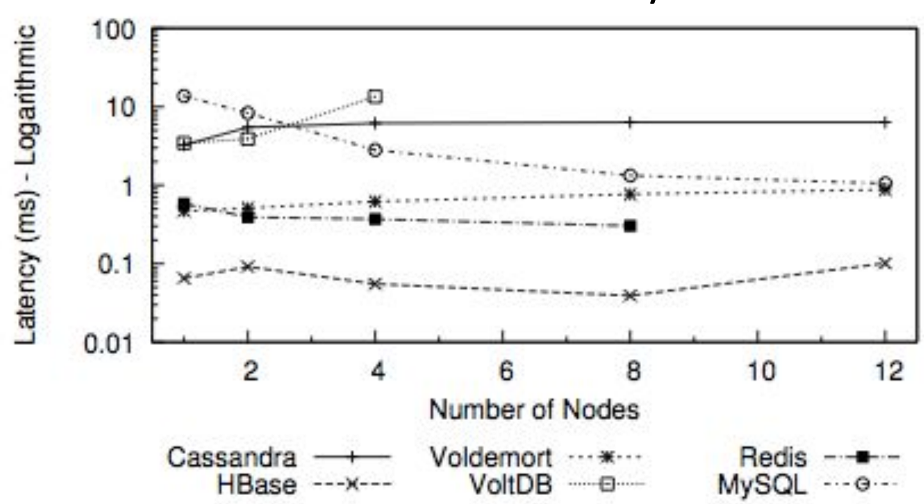
Throughput



Read Latency



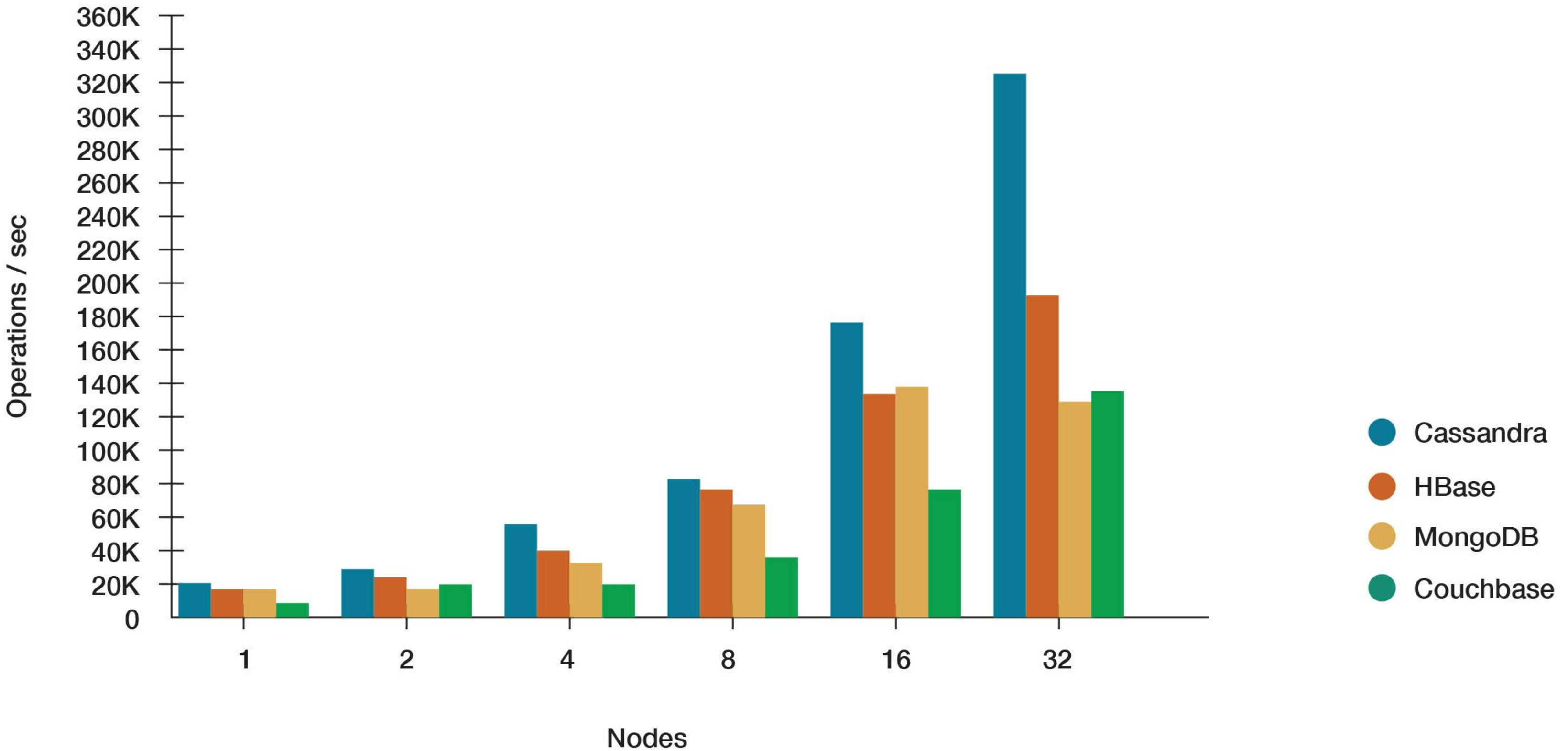
Write Latency



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

https://www.datastax.com/nosql-databases/benchmarks-cassandra-vs-mongodb-vs-Hbase



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?