Poster ID: 3

An Empirical Evaluation of Distributed Key/Value Storage Systems

A20358448

Mr. Nikhil Brahmankar

More about NoSQL Databases...

As the popularity of data virtualization continues to rise, companies are increasingly relying on data storage and retrieval mechanisms like NoSQL to extract tangible value out of the voluminous amounts of data available today. Emerging technologies like Big Data and Cloud Computing have driven the adoption of NoSQL technology.

Key Characteristics of NoSQL Databases

1. Distributed Computing (Scalability, Reliability, Sharing of Resources, Performance)
2. More Flexible Data Model (Key Value Stores, Document Stores, Column Based Stores, Graph Databases, XML Databases)
3. Asynchronous Inserts & Updates/Weak Transactional
4. Follows BASE (Basically Available, SoftState, Eventual Consistency) instead of ACID (Atomicity, Consistency, Isolation, Durability)
5. Query Language
6. No Joins
7. Low Cost
8. Easy Implementation
9. Good for scenarios which mostly require querying/searching (but not complex search or analytics) and very few or no updates.

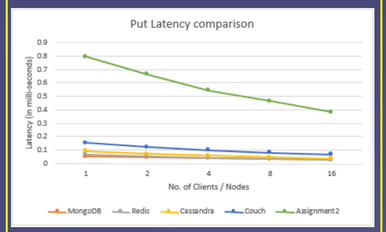
Store	Name	API	Protocol	Query Method	Written In
Key Value	Riak	Json	REST	MapReduce	Erlang
Key Value	MemcachedDB	C, Python	Memcache protocol	Memcache pattern	C, Python
Column	HBase	Java	Any Write Call	MapReduce	Java
Column	Casandra	CQL and Thrift	Thrift	Casandra query language	Java
Document	MongoDB	BSON	C	Dynamic object based language & MapReduce	C++
Document	CouchBase	Mem-cached	Memcached REST interface for cluster configuration	Javascript	C, C++, Erlang
GraphBase	Info Grid	Java	OpenID, RSS, Atom, JSON, Java embedded	Web user interface with HTML, RSS, Atom, JSON output, Java native	Java
GraphBase	Infinite Graph	Java	Direct Language Binding	Graph Navigation API, Predicate Language Qualification	Java

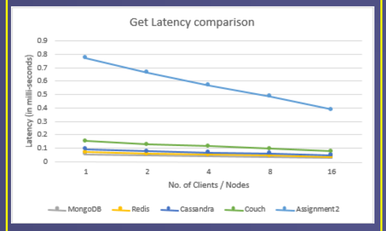
NoSQL Advantages	NoSQL Disadvantages
High Scalability	Too many options (Above 150), which one
Schema Flexibility	Limited query capabilities (so far)
Distributed Computing (Reliability, Scalability, Sharing of Resources, Speed)	Eventual consistency is not intuitive to program for strict scenarios like banking applications.
No complicated relationships	Lacks Joins, Group by, Order by facilities
Lower cost (Hardware Costs)	ACID transactions
Open Source – All of the NoSQL options with the exceptions of Amazon S3 (Amazon Dynamo) are open-source solutions. This provides a low-cost entry point.	Limited guarantee of support – Open source

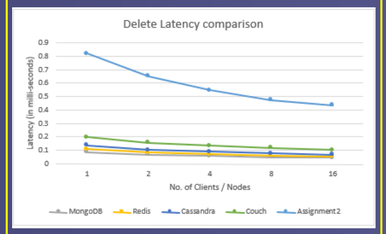
Experiment Setup

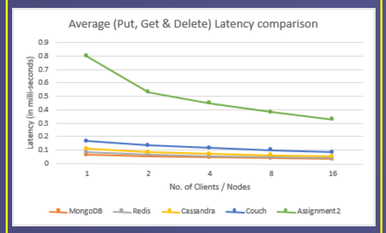
- Commercial Cloud
  - Amazon EC2
  - M3. Medium (16 nodes)
- Micro benchmark Settings
  - Each node having it's separate MongoDB, Redis, Cassandra, and CouchDB database instances
  - N clients to N Servers: All to All communication pattern
  - Randomly generated key—value pairs with optional headers: 2 byte header, 10 byte keys and 90 bytes values

Latency Comparison

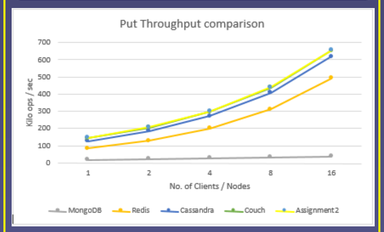


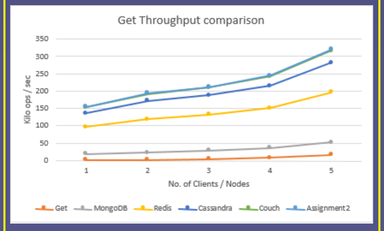


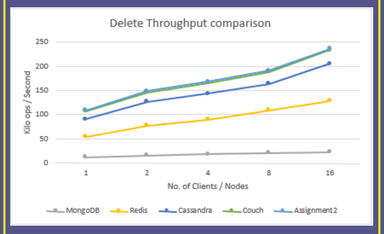


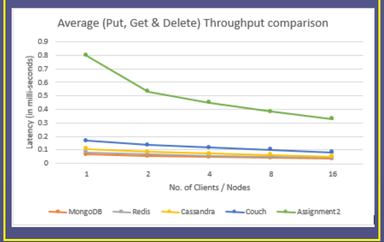


Throughput Comparison

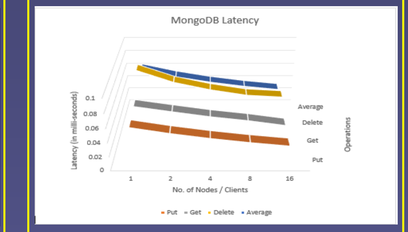


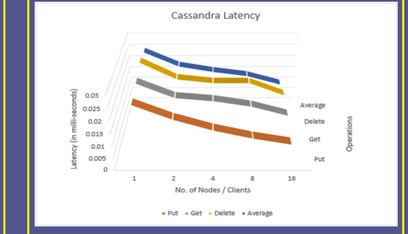


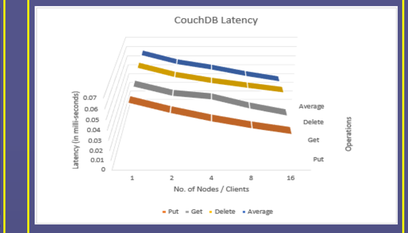


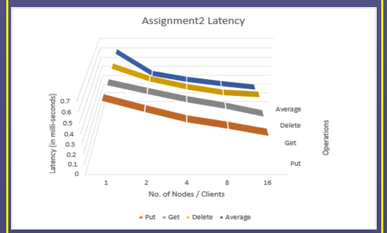


Individual System Latency

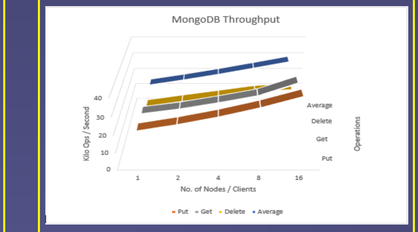


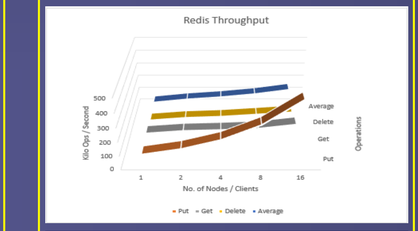


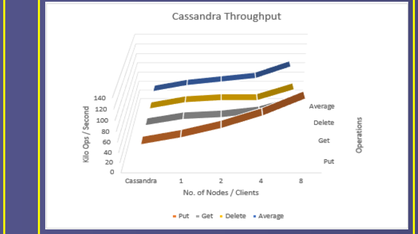


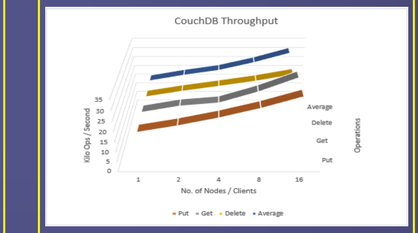


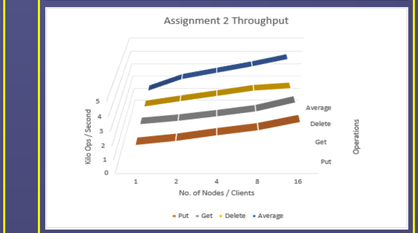
Individual System Throughput











Methodology

⇒ Each node acts as Client and Server both

⇒ Server accepts the data from various Clients using Socket and Multithreading concepts.

⇒ As soon as Client connects to Server for the first time, there is a new thread created on Server and this newly created thread is then taken over by the Server thread handler in order to fulfill Client's request. The connection between Client and Server is released only after all the client operations are completed or client process is terminated. This helps to reduce the overhead cost of opening and closing the connection for every small chunks of data.

⇒ Once the Server received the Key—Value pair data form Client node, it extracts the first two bytes / characters to confirm the type of database and operation

(e.g. 01 — Put into Concurrent Hash Map,
02 — Get from Concurrent Hash Map,
03 — Delete from Concurrent Hash Map,
11 — Put into MongoDB,
12 — Get from MongoDB,
13 — Delete from MongoDB, etc.)

⇒ Based on the above inputs, the operation is performed and the output is sent back to Client

⇒ The status received by Client is either stored into log file or displayed on the screen

Conclusion

Here is another quick comparison between NoSQL and RDMS:

Opt NoSQL
1. If data is huge, unstructured, sparse/growing
2. Less rigid schema
3. Performance & Availability preferred over Redundancy
4. While scaling out is an out-of-the-box feature, it does not prevent scale up,
5. Cost Effective- uses clusters of cheap commodity servers to manage the exploding data and transaction volumes

Opt RDBMS
1. If Analytics, BI or Reporting is required.
2. For Benefits of ACID
3. Rigid Schema
4. No redundancy allowed
5. Allows Scale up & limited Scale-out (sharding)
6. Expensive- rely on expensive proprietary servers and storage systems

1. Redis

Highlight: Performance **License:** BSD

Disk-backed in-memory database, Dataset size limited to computer RAM (but can span multiple machines' RAM with clustering), Master-slave replication, automatic failover, Simple values or data structures by keys, Bit operations, Has sets (also union/diff/inter), Has lists (also a queue; blocking pop), Has hashes (objects of multiple fields), Sorted sets (high score table, good for range queries), Has transactions, Values can be set to expire (as in a cache), Pub/Sub lets one implement messaging

Best used: For rapidly changing data with a foreseeable database size (should fit mostly in memory). (e.g.: To store real-time stock prices. Real-time analytics. Leaderboards. Real-time communication. And wherever you used memcached before)

2. Cassandra

Highlight: Store huge datasets in "almost" SQL **License:** Apache

CQL3 is very similar SQL, but with some limitations that come from the scalability (most notably: no JOINs, no aggregate functions.), Querying by key, or key range (secondary indices are also available), Tunable trade-offs for distribution and replication, Data can have expiration (set on INSERT)., Writes can be much faster than reads (when reads are disk-bound), Map/reduce possible with Apache Hadoop, Very good and reliable cross-datacenter replication, Distributed counter datatype., can write triggers in Java.

Best used: When you need to store data so huge that it doesn't fit on server, but still want a friendly familiar interface to it (e.g.: Web analytics, to count hits by hour, by browser, by IP, etc. Transaction logging. Data collection from huge sensor arrays)

3. MongoDB

Highlight: Retains some friendly properties of SQL. (Query, index)

License: AGPL (Drivers: Apache)

Master/slave replication (auto failover with replica sets), Sharding built-in, Queries are javascript expressions, Run arbitrary javascript functions server-side, Better update-in-place than CouchDB, Uses memory mapped files for data storage, Performance over features, On 32bit systems, limited to ~2.5Gb, Text search integrated, GridFS to store big data + metadata (not actually an FS), Has geospatial indexing, Data center aware

Best used: If you need dynamic queries. If you prefer to define indexes, not map/reduce functions. If you need good performance on a big DB. If you wanted CouchDB, but your data changes too much, filling up disks (e.g.: For most things that you would do with MySQL or PostgreSQL, but having predefined columns really holds you back)

4. CouchDB

Highlight: DB consistency, ease of use **License:** Apache

Bi-directional replication, continuous or ad-hoc, with conflict detection, thus, master-master replication, write operations do not block reads, Previous versions of documents are available, Crash-only (reliable) design, Needs compacting from time to time, Views: embedded map/reduce, Formatting views: lists & shows, Server-side document validation possible, Authentication possible, Real-time updates via '_changes'

Best used: For accumulating, occasionally changing data, on which pre-defined queries are to be run. Places where versioning is important (e.g.: CRM, CMS systems. Master-master replication is an especially interesting feature,