ILLINOIS INSTITUTE OF TECHNOLOGY

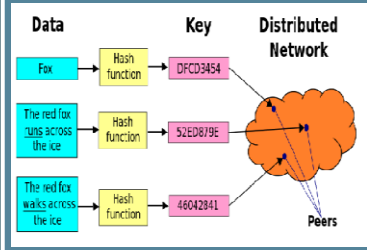
An Empirical Evaluation of Distributed Key/Value Storage Systems

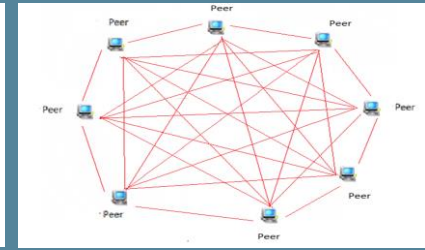
Poster Id : 06 CWID : A20352720, Vishwanath Bidari ,Computer Science CWID:A20352720,
Instructor: Ioan raicu

Abstract :

- A distributed hash table (DHT) is a class of a decentralized distributed system that provides a lookup service similar to a hash table: (key, value) pairs are stored in a DHT, and any participating node can efficiently retrieve the value associated with a given key.
- The exponential growth of data has paved way for the Distributed storage systems. The possibility of storing huge data of various forms (e-mail, photos, videos, logs etc) on a single large disk was a distant reality as it is impossible to store everything on a single disk. Also disk failure posed a major challenge. As a result, researchers and developers came up with an idea of building distributed storage systems by storing data across multiple disks on different nodes.

Architecture of DHT





Motivation :

- **Autonomy and decentralization**: the nodes collectively form the system without any central coordination.
- **Fault tolerance**: the system should be reliable (in some sense) even with nodes continuously joining, leaving, and failing.
- **Scalability**: the system should function efficiently even with thousands or millions of nodes.
- DHTs must deal with more traditional distributed systems issues such as load balancing, data integrity, and performance

This project compare above characteristics of different Key storage systems.

Proposed Work :

- This project implemented DHT with different Key/Value storage systems in Amazon AWS service using ec2 instances. The different systems used are MongoDB, Redis, CouchDB, Cassandra.
- This project evaluate various distributed key/value storage systems and calculate latency, throughput of each operation insert/lookup/remove separately, and as an average across all 3 operations.
- Latency presents the time per operation (insert/lookup/remove).
- Throughput: The number of operations (insert/lookup/remove) the system can handle over some period of time, measured in Kilo Ops/s
- Compare the above storage systems to local system implemented for DHT.

1. Redis

- **Written in**: C,
- **Main point:** Blazing fast(BSD)
- **Protocol:** Telnet-like, binary safe
- **Best used**: For rapidly changing data with a foreseeable .
- **For example**: To store real-time stock prices. Real-time analytics. Leaderboards. Real-time .

2. Cassandra (2.0)

- **Written in**: Java
- **Main point:** Store huge datasets in "almost" SQL (Apache)
- **Protocol**: CQL3 & Thrift.
- **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. **For example:** Web analytics, to count hits by hour, by browser, by IP, etc. transaction logging.

3. MongoDB (2.6.7)

- **Written in**: C++
- **Main point**: Retains some friendly properties of SQL.
- **Protocol**: Custom, binary (BSON)
- **Best used:** If you need dynamic queries. If you prefer to define indexes, not ap/reduce functions. **For example**: For most things that you would do with MySQL or PostgreSQL, but having redefined columns really holds you back

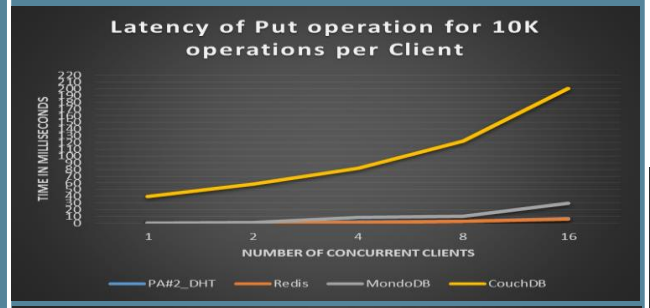
4. CouchDB (V1.2)

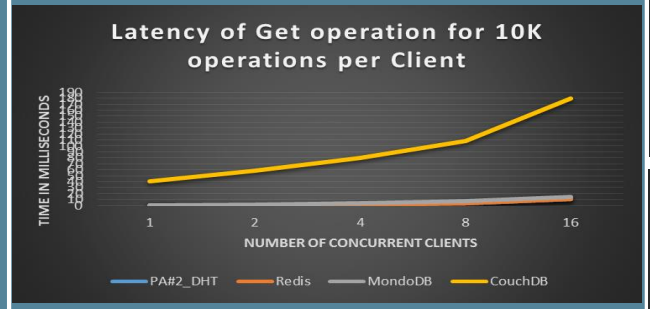
- **Written in:** Erlang
- **Main point:** DB consistency, usage
- **Protocol:** HTTP/REST
- **Best used:** For accumulating, occasionally changing data, on which pre-defined queries are to be run.
- **For example:** CRM, CMS systems. Master-master replication is an especially interesting feature.

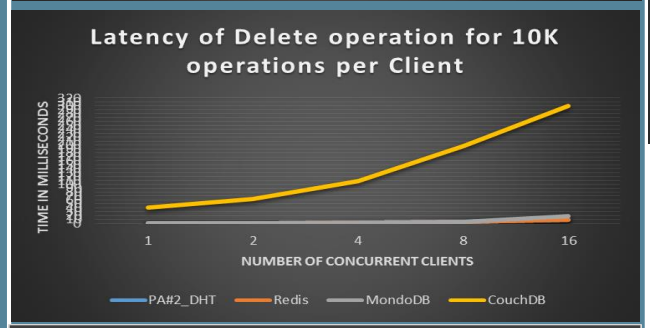
Experimental set up:

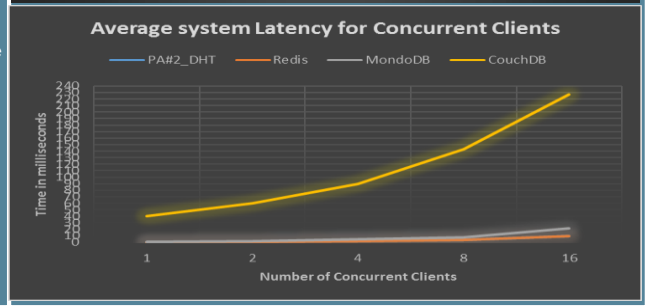
- Use Amazon EC2 service
- Launch 16 m3.medium instances
- Randomly generated Key: 10 byte, value:90 byte , Workload : 100 k

Evaluation : 1. Latency graphs : x-axis(nodes) y-axis (time in ms)

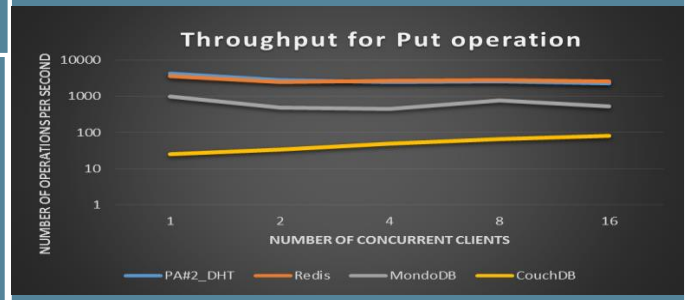


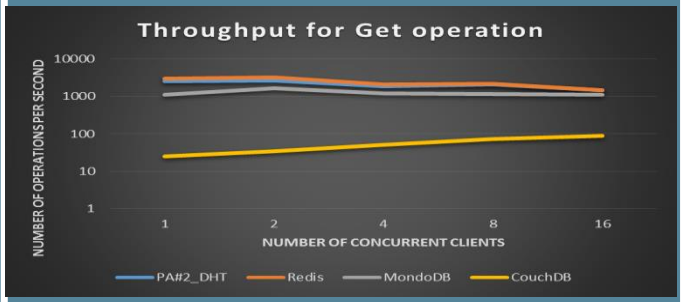


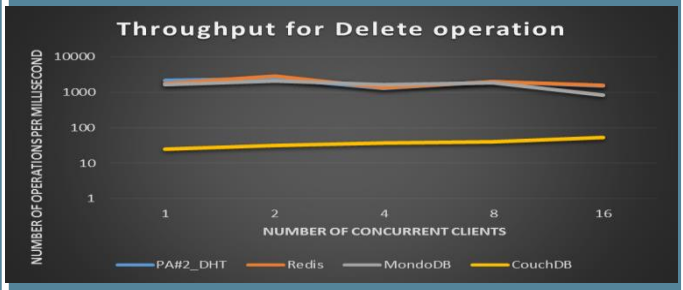


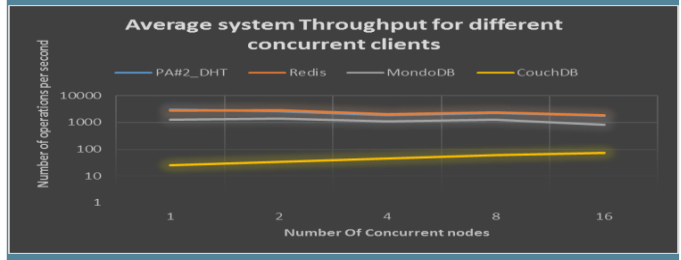


Evaluation : 2. Throughput graphs : x-axis(nodes) y-axis (K ops/sec)









Conclusion :

Redis and DHT systems are faster compared to other systems because memory based. MongoDB and Cassandra good for large data. CouchDB is good for versioning.

Reference :

https://www.mongodb.org, http://cassandra.apache.org
http://couchdb.apache.org, http://redis.io