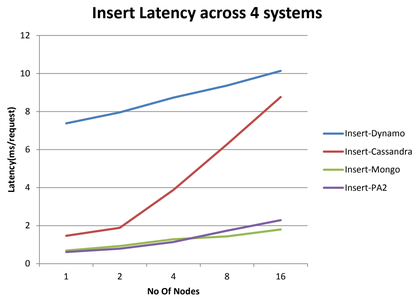
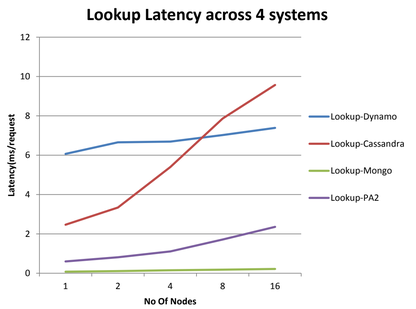
POSTER ID:102

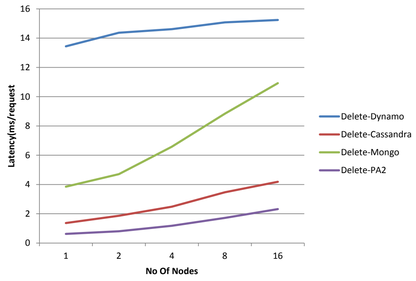
A Comparison of Distributed Databases With a Distributed Key/Value Store

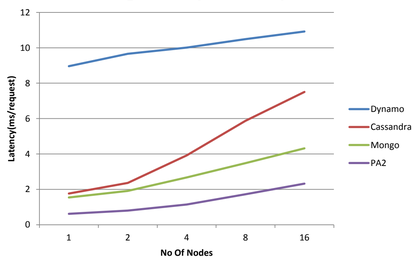
Done by Shivakumar Vinayagam
A20341139 svinayag@hawk.iit.edu

ILLINOIS INSTITUTE OF TECHNOLOGY









Our world is being dramatically changed by the advent of distributed systems on the scale of the Internet. Recently, such distributed systems concepts as cloud computing, Map-Reduce /Hadoop, online Web services, peer-to-peer networks, distributed networked games, content delivery networks, etc. Hence Distributed systems are becoming more common in use and are replacing traditional systems like SQL,etc. Here we perform a comparison between 3 Distributed database systems namely, DynamoDB ,Cassandra and MongoDB with the Distributed key/value store built as a part of Assignment 2.

A ***Distributed Database*** is a database in which storage devices are not all attached to a common processing unit such as the CPU, and which is controlled by a distributed database management system (together sometimes called a distributed database system). It may be stored in multiple computers, located in the same physical location; or may be dispersed over a network of interconnected computers. Unlike parallel systems, in which the processors are tightly coupled and constitute a single database system, a distributed database system consists of loosely coupled sites that share no physical components.

Amazon DynamoDB is a fast and flexible NoSQL database service for all applications that need consistent, single-digit millisecond latency at any scale. It is a fully managed cloud database and supports both document and key-value store models. DynamoDB exposes a similar data model and derives its name from Dynamo (an internal storage system used initially for their own website Amazon.com), but has a different underlying implementation. Dynamo had a multi-master design requiring the client to resolve version conflicts and DynamoDB uses synchronous replication across multiple datacenters for high durability and availability.

Apache Cassandra is an open source distributed database management system designed to handle large amounts of data across many commodity servers, providing high availability with no single point of failure. Cassandra offers robust support for clusters spanning multiple datacenters, with asynchronous masterless replication allowing low latency operations for all clients. Cassandra is essentially a hybrid between a key-value and a column-oriented (or tabular) database. Its data model is a partitioned row store with tunable consistency. Rows are organized into tables; the first component of a table's primary key is the partition key; within a partition, rows are clustered by the remaining columns of the key. Other columns may be indexed separately from the primary key. Tables may be created, dropped, and altered at run-time without blocking updates and queries. Cassandra does not support joins or subqueries. Rather, Cassandra emphasizes denormalization through features like collections.

Distributed Key/Value Store (Assignment 2).The peer contains three parts. The first part binds to the server port and listens on it for incoming connections. This part creates a new thread for each connection which it receives and handle of the handling of the msg to it. The started up thread now reads the message and check whether it is either add entry, delete entry, get entry or duplicate entry. Depending on the type of message it does the corresponding work which is need and sends any data or ACK back to the peer client part which sent the request. Second part is the resilience thread which can be either started up or not based on the initializing values for the program startup. The thread if started initially sleeps for 60 seconds in order for the local hash table to build up and also the various key/values to be added to it. Then after 60 secs, it read the local Distributed hash table (DHT) and finds any keys which were newly added after the previous call. This then calls duplicate to a peer to the right of this peer in the peer table. This duplicate call is sent to the respective peer server listener which starts a thread for this call. This duplicate call in turn adds this value to another local Hash table which contains the duplicates values alone. Thus this thread keeps running in an endless loop till the server is shut down. The Third part is the client part of the peer. The peer client thread part either starts up the test case function or the User IO interface function depending on the value of either True or False set on program startup. If the test case function is called, it executes the 10000 add, lookups and deletes, to give the results. If the user interface is called it gives a prompt of options to the user on the command line prompt and then based on the option called calls either of the add entry, delete entry, get entry or print local DHT.

MongoDB (from humongous) is a cross-platform document-oriented database. Classified as a NoSQL database, MongoDB eschews the traditional table-based relational database structure in favor of JSON-like documents with dynamic schemas (MongoDB calls the format BSON), making the integration of data in certain types of applications easier and faster. Released under a combination of the GNU Affero General Public License and the Apache License, MongoDB is free and open-source software. MongoDB scales horizontally using sharding. The user chooses a shard key, which determines how the data in a collection will be distributed. The data is split into ranges (based on the shard key) and distributed across multiple shards. (A shard is a master with one or more slaves.) MongoDB can run over multiple servers, balancing the load and/or duplicating data to keep the system up and running in case of hardware failure. Automatic configuration is easy to deploy, and new machines can be added to a running database.

Experiment Setup:

The evaluation of the various systems was performed in Amazon AWS Cloud, using EC2 Instance of type M3.Medium with the no. of nodes ranging from 1 to 16 instances. Hence Strong scaling was performed by increasing the no. of listening servers as the no of clients or operations increase. Hence there is a 1:1 mapping between the clients and servers. Also the No of operation per client is fixed at 10K operations.

We perform the operations of insert, lookup and delete on 10K entries for each database system. We also use parallel ssh to start the various client and servers simultaneously. Also we have scripts to store the values of the time taken in order to get the data after the operations are performed.

Evaluation:

There are 8 graphs displayed 4 for latency and 4 for throughput .The Time taken was calculated for 10k operations of insert ,lookup ,then delete and the latency and throughput were calculated for each range of nodes. The average was calculated across the operation per system per range of nodes and then plotted. As can be seen in the graphs, the latency of the dynamo DB is the worst .This is due the limitation on the read and write throughput which is being placed on the Table .Hence even though the network latency might be low, the processing latency is quite high as there is a limitation on the no. of operation which can be done at any time. This also results on a low throughput also for dynamo DB. Also the Cassandra uses SQL format to store and retrieve the data . This results in a higher latency as the no of operations increases due to the structure of the table. An this performs poorly in comparison to the other data bases since they are key/value stores or in other words NoSQL data stores. In comparison ,the distributed key/value store designed as part of assignment, does better than average in most cases since it is just a bare metal implementation. It does not have any Complex replication or state transition algorithm to ensure its reliability or security. It simply uses socket and TCP to perform the operations and also has a few features to duplicates keys. Mongo DB on the other end performs best for insert and lookup since it is a Hash table but does the worst in delete since that is a heavy process. Hence on average , based on latency our system does the best, but based on throughput Mongo DB does the best as it averages well due to lookup.

Conclusion:

Hence we can conclude the Distributed Key/value Store does well on average case across all operations and measurements as it is just a simple bare metal implementation. But average throughput wise mongo dB does Best, so on the whole we can say mongo dB is the best except in case of delete operation where it does badly. Dynamo DB is performing poorly due to it limitation set by amazon. And Cassandra does well for a SQL data store in comparison with noSQL data stores.