A Convergence of Key-Value Storage Systems from Clouds to Supercomputers

CS554 – Data Intensive Computing CWID – A20402686

Summary of the paper

This paper presents ZHT (Zero-hop distributed key-value store system) which is used in high performance computing. In current high-performance systems, storage is separated from compute systems they work through network. But it creates many problems like, less concurrency and more latency. Also, metadata management has become major problem on large scale systems. Cloud based distributed systems are facing storage problems. Cloud applications need to handle dynamic nature of nodes (join and leave). To handle all this problem, ZHT system has been built (instance of NoSQL database).

ZHT has features like light-weight, dynamically allows nodes to join and leave, fault tolerance and consistent hashing mechanism. ZHT also implemented new operations like append, compare-swap and callback. ZHT can be scaled up to 32K cores with 1.1ms latency 18M operations/sec throughput.

ZHT requires dynamic membership management. The node ID is randomly distributed in network. A long string is mapped to index value by hash function which can be used to get communication address like IP address and port number. Entire membership table requires less than 1% of memory of each node.

There are many primitive operations like insert, lookup, remove, append, cswap (compare and swap) and callback operation. ZHT architecture is consisting of Physical node (physical machine which runs several ZHT instances), Instance which is a server process which handles request from client, Partition which is continuous range of key address space and Manager which is a service to take charge on starting and shutting down of ZHT, managing membership table and partition migration.

Dynamic membership includes dynamically joining and leaving nodes. If one node wants to migrate, partitions of that node are migrated as a whole, instead of migrating individual key-value pairs. When any node joins a network, it checks out membership table from ZHT manager of random physical node. At the time of node departure, two cases are possible i.e. planned node departure and unplanned node departure. In client-side state, client updates membership table by requesting server. ZHT has event-driven server architecture. ZHT uses data replication to keep data persistent. ZHT allows user to access only one replica of data for write operation. ZHT introduces new technique called Non-Volatile Hash Table (NoVoHT) which uses log based persistence mechanism with periodic checkpointing. ZHT is implemented in C or C++ with few dependencies.

The performance evaluation of ZHT is done by synthetic benchmarks compared with systems such as Linux cluster, Amazon EC2 cloud, IBM Blue Gene/P supercomputer, HEC cluster, DataSys, Fusion and Kodiak. This evaluation is done on metrics like latency, throughput, idle throughput, scalability and efficiency.

ZHT is used in applications like FusionFS, IStore, MATRIX, Slurm++ and Fabriq.

How is this work different than the related work?

In ZHT, three new features are added in primitive operations, like append, compare and swap and callback along with primitive operations like insert, lookup and remove. Cassandra system adopts a logarithmic routing algorithm where ZHT uses constant routing algorithm. Also, in ZHT, there are both static and dynamic membership available where other systems have only static membership available. Scalability is more for ZHT as compared to that of DynamoBD. There are many systems that uses distributed hash tables and key-value stores, but ZHT uses Non-Volatile Hash Table (NoVoHT) which has a log-based persistence mechanism with periodic checkpointing. ZHT system's throughput is better than Memcached on BLUE GENE/P. DynamoDB Running cost is 65 times more than ZHT on 2-node scale.

Identify the top 3 technical things this paper does well.

- Metadata Management.
- Dynamically allows node to join and leave.
- Non-Volatile Hash Table for data persistence.
- New operations like append, swap & compare and callback operation.

Identify 3 things the paper could do better.

- Method to increase the efficiency of ZHT when number of node increases.
- Method to decrease the latency of ZHT when number of node increases as ZHT's latency is more as compared to that of Memcached.
- ZHT could use JAVA programming language for more features.

If you were to be an author of a follow up paper to this paper, what extensions would you make to improve on this paper?

If I was the author of this paper, I would have added method to improve efficiency and decrease the latency of ZHT when number of node increases.