Summary

This paper narrows down the work of ZHT (Zero Hop Distributed Key Value Storage Systems) implementation from cloud to supercomputer. This paper presents several real systems that have adopted ZHT, namely FusionFS (a distributed file system), IStore (a storage system with erasure coding), MATRIX (distributed scheduling), Slurm++ (distributed HPC job launch), Fabriq (distributed message queue management).

Today's high-end computing systems are generating datasets that are increasing in both complexities and volume. Increasing demands form the clients for a more reliable and resilient systems are at its peak. Metadata and file movement between the nodes, I/O throughput, cost cutting of hardwares in extreme scale system, efficiency etc. are some of the few bottlenecks face by the current state of the art system (e.g. Exascale systems 10^{18} ops/sec). These problems raise a serious concern for future distributed systems. To overcome all the challenges mentioned above ZHT is proposed.

As an initial attempt to meet these needs the paper comprehensively discussed about ZHT which is an instance of NoSqlDatabase. Researchers have claimed that ZHT has been tuned to meet the specific requirements for high- end computing i.e. trustworthy/reliable hardware, fast networks, nonexistent" churn", low latencies. Unlike other conventional distributed hash table and key value stores ZHT implements important key features like light-weight, dynamically allowing nodes to depart and join, fault tolerant through replication, handling failures gracefully, efficiently propagating events throughout the system, a customized consistent hashing mechanism. Apart from these salient features, some new operations have been implemented via ZHT in distributed computing i.e. append, compare swap/ and state change _ callback in addition to insert/lookup/remove. Scalability and performance of ZHT has been evaluated up to 8k-node and 32k instances. On the 32K-core scale we achieved more than 18M operations/sec of throughput and 1.1ms of latency at 8K-node scale.

This paper also clearly states the working of ZHT in large scale computing. Implementation of dynamic membership management is proposed for large scale systems as It would be highly beneficial for environments like cloud computing since nodes can join (to increase the performance) or leave (node failure or maintenance). All the node ID space and membership are treated as a ring- shaped name space and are distributed throughout the network. A hash function maps arbitrarily long strings to index values, which can then be used to efficiently retrieve the communication address (e.g. host name, IP address etc.) from a local in-memory membership table. Each physical node in ZHT has one manager, holds partitions, with each partition storing key-value pairs and ZHT instances serving requests. Partition can be move across the nodes in case of node joins or fails. Paper finally evaluates the performance of ZHT under various systems ranging from Linux cluster with 64 nodes Amazon EC2 virtual cluster up to 96-nodes, to an IBM Blue Gene/P supercomputer with 8K-nodes. ZHT offers superior performance for the features and portability it supports.

How is this work different than the related work

In the past few years many distributed hash tables and key/value storage systems have been proposed and implemented. Some of them are widely discussed and researched are Chord, CAN , Pastry , Kademlia , Tapestry , RIAK and Cassandra) adopt logarithmic routing algorithms, resulted in increased latency with systems scale. Some other works such as Dynamo and Memcached , use constant routing algorithms to achieve a nearly constant latencies like ZHT. Most of these key/value storage techniques do not use dedicated managers like ZHT to trigger an event or to move the file from one node to other. DHT is a widely used hash table but it does not provide any guarantee of consistency and integrity and it is not reliable for group queries also.

Amazons's Dynamo is key-value storage system that Daynamo hosts some of the core services. It started the trend of NoSQL and claimed to be the a Zero- Hop DHT where each server has enough routing information locally to route requests to the appropriate target server directly. Memcached is also an efficient Key value store. It was designed as a cache to speed up distributed data access such as web page caches. Due to its specific purpose, it does not support dynamic membership, replication and persistence. The length and the key values are strictly limited to 250 and 1 megabytes respectively.

Cassandra is like Amazon's Dynamo and very popular in industries, but it gets little use in high performance computing areas as many supercomputers don't have good support on Java stack. One of the major drawback of Cassandra is its logarithmic routing strategy, which makes the performance scalability a big issue. HyperDex is also a distributed key value store that focuses on boosting performances. SkimpyStash adopts new backed technology to boost the performance and uses a hash table directory in RAM to index key-value pairs stored in a log-structured manner on flash. Masstree functions by keeping all data in memory in a form of concatenated B+ trees. It can execute more than six million simple queries per second. LOCUS is a system equipped customize SSD design. Main motivation of LOCS was to overcome inefficiencies to naively combining LSM-tree based KV stores with SSD. SILT focuses on using algorithmic and systemic techniques to balance the use of memory, storage and computation.

Identify the top 3 technical things this paper does well

- This paper has given the detailed view of ZHT and why it is important for distributed systems like cluster computing, cloud computing etc.
- Comparison between other key/value storage systems with ZHT is shown to prove the superiority over other key/value storage systems. Thus, proving the main objective of the paper.
- The paper has clearly explained all the operations of ZHT like insert, lookup, remove, append, cswap, callback. The working of all the operations are clearly stated with challenges they face with the existing systems and what are the improvements ZHT has provided.

Identify 3 things the paper could do better

- The paper could have compared the migration of files and metadata management over the nodes in the known file systems like HDFS, GFS, NFS etc. in detail using DHT (Distributed Hash Table).
- Performance or throughput of the I/O under ZHT is hardly discussed in the paper.
- Paper could have provided more information on the types of storage systems ZHT uses to compare throughput, latencies, fault tolerance of the storage disk.

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?

As an author of the paper I would have included the metadata management and file migration over the nodes in HDFS, GFS, NFS, PVFS to compare the results between DHT and ZHT.