Exploring Distributed Hash Tables in High-End Computing

Tonglin Li
Computer Science
Illinois Institute of Technology
Chicago, IL, USA
tonyli@hawk.iit.edu

Raman Verma
Computer Science
Illinois Institute of Technology
Chicago, IL, USA
rverma3@iit.edu

Xi Duan
Computer Science
Illinois Institute of Technology
Chicago, IL, USA
xduan@iit.edu

Hui Jin
Computer Science
Illinois Institute of Technology
Chicago, IL, USA
hjin6@iit.edu

Ioan Raicu
Computer Science
Illinois Institute of Technology
Chicago, IL, USA
iraicu@cs.iit.edu

ABSTRACT

Over the last decade, storage systems have experienced a 10-fold increase between their capacity and bandwidth. This gap is predicted to grow faster with exponentially growing concurrency levels, with future exascales delivering millions of nodes and billions of threads of execution. A critical component of future file systems for high-end computing is metadata management. This extended abstract presents ZHT, a zero-hop distributed hash-table, which has been tuned for the specific requirements of high-end computing. The primary goal of ZHT is excellent availability, fault tolerance, high throughput, and low latencies.

1. INTRODUCTION

The current architecture of high-end computing (HEC) systems is decades-old and has persisted as we scaled from gigascales to petascales. In this architecture, storage is completely segregated from the compute resources and are connected via a network interconnect. This approach will not scale several orders of magnitude in terms of concurrency and throughput, and will thus prevent the move from petascales to exascale. One of the major bottlenecks in current state-of-the-art storage systems is metadata management. Metadata operations on parallel file systems can be inefficient at large scale. Early experiments on the BlueGene/P system at 16K-core scales shows the various costs (wall-clock time measured at remote processor) for file/directory create on GPFS [1]. Ideal performance would be to have a flat line. If care is not taken to avoid lock contention, performance degrades rapidly, with operations (e.g. create directory, create file, etc) that took milliseconds on a single core, taking over 1000 seconds at 16K-core scales. [3, 4]

ZHT has been tuned for the specific requirements of high-end computing (e.g. trustworthy/reliable hardware, fast networks, non-existent "churn", low latencies, and scientific computing data-access patterns), and we believe it is a good abstraction to build distributed metadata management for distributed file systems.

2. RELATED WORK

DHTs have an important role in building support for scalable meta-data across extreme scale systems. Some of the DHTs from the literature are Kademlia[11], CAN[12], Chord [13], Pastry[14], Tapestry[15], Memcached[16], Dynamo[2], Cycloid[17], Ketama[18], RIAK[19], Maidsafe-dht[20], and C-MPI[8]. It is important to point out that several key features of traditional DHTs are not necessary in HEC. Most HEC environments are batch oriented, which implies that a system that is configured at run time, generally has information about the compute and storage resources that will be available. This means that the amount of resources (e.g. number of nodes) would not change dynamically, and the only reason to decrease the allocation is either to handle failed nodes, or to terminate the allocation. Furthermore, nodes in HEC are generally reliable and have predicable uptime (nodes start on allocation, and nodes shut down on deallocation). This implies that node "churn" in HEC is virtually non-existent. HEC systems are generally locked down from the outside world, behind login nodes and firewalls, and although authentication and authorization is still needed, full communication encryption is wasteful for a large class of scientific computing applications that run on many HEC systems.

There has been some uptake recently in using traditional DHTs in HEC, namely the C-MPI [8] project, in which the Kademlia DHT has been implemented and shown to run well on 1000 nodes on a SiCortex machine. Another recent project using DHTs on a HEC is DataSpaces [7], which deploys a DHT on a Cray XT5 to coordinate in-memory data management for simulation workflows. Both C-MPI and DataSpaces fail to pay careful attention to latency by ensuring constant time operations. Furthermore, they do not decouple the metadata from the data management, causing potentially poor data locality. Amazon Dynamo [2] is arguably similar to the proposed ZHT, also claiming to be a zero-hop distributed hash-table. However, it's a non-open-source project, which makes its adoption not possible.

3. ZHT: A ZERO HOP DISTRIBUTED HASH TABLE

We propose a new data-structure, named ZHT (Zero hop distributed Hash Table), which has been simplified and tuned for the specific requirements of High-End Computing (HEC). The ZHT features are described in this section.

Hash Functions: ZHT uses the SDBM hash function, due to its simple implementation, consistency across different data types (especially strings), and efficient performance.

Membership Table: The proposed hash functions map an arbitrary long string directly to an index value, which can then be used to efficiently retrieve the communication address (e.g. network name, IP address, MPI-rank) from a membership table.

Failure Handling: ZHT gracefully handles failures, by lazily tagging nodes that do not respond to requests repeatedly as failed (using exponential backoff). Once nodes are marked as down, they are assumed never to return until ZHT is restarted.

Replication: ZHT uses replication to ensure data stored persists during failures. New data created is pro-actively replicated to nodes in close proximity in the membership ID space. Ideally, once we support the network-aware topology of node ids (part of the future work section), the replication will consume the least amount of shared network resources by communicating only with neighbors in close proximity, and therefore improving scalability at extreme scales. We implemented replication on sever side. When requests (insert and remove) are sent to servers, a thread will send the same request to corresponding replicas asynchronously. This will certainly introduce some overhead due to the sharing resource (CPU and network bandwidth). But our implementation don't introduce too much extra overhead when increase the number of replicas.

Persistence: ZHT is a distributed in-memory data-structure. We use a light-weight persistent hashtable with Kyotocabinet [6], which stores the ZHT state to persistent storage in real time.

Implementation: The application programming interface (API) of ZHT is kept simple and follows similar interfaces for hash tables. The three operations that ZHT currently supports are 1. `bool insert(key, value)`; 2. `value lookup(key)`; 3. `bool remove(key)`. Another operation `bool broadcast(key, value)` will be supported soon. Replication occurs asynchronously between the hashed destination and its neighbors. The lookup operation would return the value from ZHT, if it existed. The remove operation would remove the value with the associated key. The broadcast operation would transmit the key/value pair over the edges of the spanning tree or gossip protocol with the goal to distribute the key/value pair to all the caches. ZHT uses UDP and TCP based protocols for both stateful and stateless communication.

Complex data structures are serialized into a string with the help of Google Protocol Buffers [5] and parse it back into the original structure at the receiving end. Further, the key-value pairs are stored into Kyotocabinet [6].

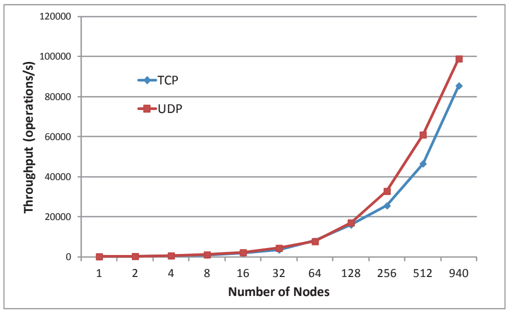

Figure 1: ZHT throughput on SiCortex

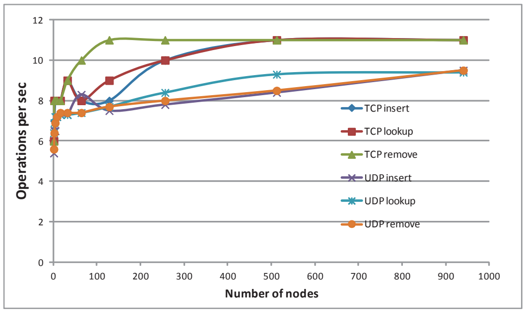

Figure 2: ZHT latency on SiCortex

4. PERFORMANCE EVALUATION

In our implementation we completed an prototype of ZHT, which supports three of the four proposed operations, namely insert/lookup/remove. Prior to engaging in implementing ZHT, we explored the feasibility of using existing DHT implementations (Tapestry, Chord, Maidsafe-dht, and C-MPI), but came to the conclusion that they are all too heavy weight for HEC, were not feature complete, or had significant dependencies to resolve. The preliminary ZHT implementation was implemented in C++, and uses a proprietary binary TCP-based and UDP-based communication protocol. We performed experiments (on the SiCortex SC5832 at ANL, 972 nodes with 5832 cores) to measure the overheads of insert/remove/lookup. Overheads were significantly lower than the DHTs that were investigated, with about 10ms per operation at modest scales of 5832-cores (as opposed to 30 ms for Chord and 1000 ms for Maidsafe-DHT at 16-core scales). Figure 1 shows the throughput for ZHT for a range of scales from 1 node (6 cores) to 940 nodes (5640 cores). Each client (1 client per node) performs 100K random 132-byte key inserts, 100K lookups, and 100K removes. The throughputs in operations per second increases near-linearly with scale, reaching 85K ops/sec at 940 nodes; the ideal throughput would have been 156K ops/sec. Also, as shown in the figure, the UDP protocol has better performance than TCP, which is near 100k ops/sec, and would might scale better in even large scale, nevertheless, based on the volatile propriety of UDP and bottleneck of network, UDP protocol in our implementation will lose performance when the network is saturated, after all, it does not hold a reliable way. Compared with the ideal linearly speed-up, ZHT hits more-than-half of the performance of it, it cannot be considered as a perfect work, however, it makes a giant progress in MTC.

Figure 2 shows the latency incurred by various operations on both TCP and UDP. TCP implementation keeps the latency within 11 ms while UDP does it with 9ms.

Additionally, we note that when the network saturates, TCP shows a better scalability and reliability. The performance differences among three basic operations (insert, lookup and remove) are very small and tend to be negligible. The time per operation starts at about 6ms at the smallest scales of 2 nodes (1 node scales is extremely fast because it has no network communication), and increases to 11ms to 12ms at the largest scale of 972 nodes. Since ZHT uses a direct 0-hop algorithm and that the majority of the overhead comes from network communication, it is not expected for the time per operation to increase significantly with larger scales.

5. CONCLUSION

The ideas in this position paper are transformative due to their departure from traditional HEC architectures and approaches, while proposing radical storage architecture changes based on distributed file systems to make exascale computing a reality. This paper addresses fundamental technical challenges that will become increasingly harder to address with existing solutions due to a declining MTTF of future HEC systems. This work will open doors for novel research in programming paradigm shifts (e.g. Many-Task Computing [3]) needed as we approach exascale.

ZHT optimized for high-end computing systems was architected and implemented as a foundation in the development of fault-tolerant, high-performance, and scalable storage systems. We performed an extensive performance evaluation of ZHT. We have measured the performance of ZHT over TCP and UDP at scales up to 5400-cores on a SiCortex SC5832 supercomputer. ZHT is an open source project, available at [9].

Our work will benefit the 'Many-Task Computing' paradigm that bridges the gap between high-throughput computing and high-performance computing, generally producing both compute-intensive and data-intensive workloads, and has been shown to contain a large set of scientific computing applications from many domains.

Our main message is that by combining lessons learned from parallel file systems and distributed file systems, along with new advances in hardware (e.g. solid state memory), we can define a new storage architecture that is optimized for future high-end computing at exascale and has the potential to deliver a viable storage architecture for future extreme scale high-end computing. The position of this paper is revolutionary as it breaks the accepted practice of segregating storage resource from computational resources, and leveraging the abundance of processing power, bisection bandwidth, and local I/O commonly found in future high-end computing systems.

Acknowledgements

This work was supported in part by the National Science Foundation grant NSF-0937060 CIF-72 and NSF-1054974. We want to thank our collaborators for the valuable help, feedback, and insight leading up to this work, namely Mike Wilde, Matei Ripeanu, Justin Wozniak, Arthur Barney Maccabe, Marc Snir, Rob Ross, Kamil Iskra.

REFERENCES

[1] F. Schmuck, R. Haskin, "GPFS: A Shared-Disk File System for Large Computing Clusters," FAST 2002

[2] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, W. Vogels. "Dynamo: Amazon's Highly Available Key-Value Store." SIGOPS Operating Systems Review, 2007

[3] I. Raicu, Z. Zhang, M. Wilde, I. Foster, P. Beckman, K. Iskra, B. Clifford. "Toward Loosely Coupled Programming on Petascale Systems," IEEE SC 2008

[4] Z. Zhang, A. Espinosa, K. Iskra, I. Raicu, I. Foster, M. Wilde. "Design and Evaluation of a Collective I/O Model for Loosely-coupled Petascale Programming", IEEE MTAGS 2008

[5] Google buffer protocol: http://code.google.com/apis/protocolbuffers/

[6] Kyotocabinet: http://1978th.net/kyotocabinet/

[7] C. Docan, M. Parashar, S. Klasky. "DataSpaces: An Interaction and Coordination Framework for Coupled Simulation Workflows", ACM HPDC 2010

[8] Justin M. Wozniak, Bryan Jacobs, Rob Latham, Sam Lang, Seung Woo Son, and Robert Ross. "C-MPI: A DHT implementation for grid and HPC environments", Preprint ANL/MCS-P1746-0410, 2010

[9] ZHT: Zero-Hop Distributed Hash Table, http://datasys.cs.iit.edu/projects/ZHT/index.html, 2011

[10] http://en.wikipedia.org/wiki/SiCortex2009

[11] P. Maymounkov, D. Mazieres. "Kademlia: A Peer-to-peer Information System Based on the XOR Metric", In Proceedings of IPTPS, 2002

[12] S. Ratnasamy, P. Francis, M. Handley, R. Karp, S. Schenker, "A scalable content-addressable network," in Proceedings of SIGCOMM, pp. 161-172, 2001

[13] I. Stoica, R. Morris, D. Karger, M.F. Kaashoek, H. Balakrishnan, "Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications", ACM SIGCOMM, pp. 149-160, 2001

[14] A. Rowstron and P. Druschel, "Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems," in Proceedings of Middleware, pp. 329-350, 2001

[15] B.Y. Zhao, L. Huang, J. Stribling, S.C. Rhea, A.D. Joseph, and J.D. Kubiatowicz. "Tapestry: A Resilient Global-Scale Overlay for Service Deployment", IEEE Journal on Selected Areas in Communication, VOL. 22, NO. 1, 2004

[16] B. Fitzpatrick. "Distributed caching with memcached." Linux Journal, 2004(124):5, 2004

[17] H. Shen, C. Xu, and G. Chen. Cycloid: A Scalable Constant-Degree P2P Overlay Network. Performance Evaluation, 63(3):195-216, 2006

[18] Ketama, http://www.audioscrobbler.net/development/ketama/, 2011

[19] Riak, https://wiki.basho.com/display/RIAK/Riak, 2011

[20] Maidsafe-DHT, http://code.google.com/p/maidsafe-dht/, 2011