ZHT

A Fast, Reliable and Scalable Zero-hop Distributed Hash Table

Tonglin Li
Illinois Institute of Technology, Chicago, U.S.A

Intro: Distributed key-value stores

- What is KVS?
- Why to use?
- Why not to use?
- Who's using it?
- Design issues

What

- A storage system
- A distributed hash table
- Spread simple structured data to multiple node
- Extremely simple API: <key, value>
  - **Read**: `lookup(string key)`
  - **Write**: `insert(string key, string value)`
  - Others: remove, append etc.

Why to use KVS?

- Simple read/write: put/get
- Small requests
- Fast requests
- Good as a building block of scalable systems
- Excellent scalability
  - Capacity
  - Performance

Why NOT to use KVS?

- Need traditional database: SQL
- No transaction
- No complex query:
  - Relational query
  - Partial key query
  - Range query

Who are using/building KVS?

- Amazon: Dynamo, DynamoDB
- LinkedIn: Voldmort
- Facebook: Memcached
- Google: LevelDB, BigTable*
- Apache: Cassandra*, Hbase*

Design issues

- Client request routing
- Partitioning: consistent hashing, mod
- Consistency model
- Membership management
- Failure handling/recovery

Background: Big problems ask for big systems

- **High performance computing systems**
  - **IBM Sequoia** in LLNL: 1.5 M cores, 1.5PB memory
- **Big data applications in business**
  - **Facebook**: 300 M photos, 500TB data per day (2013)
  - **Amazon**: 1.5 B items in retail, 0.5 M shoppers per day (2012)
  - **Google**: processing 20PB data per day (2009)

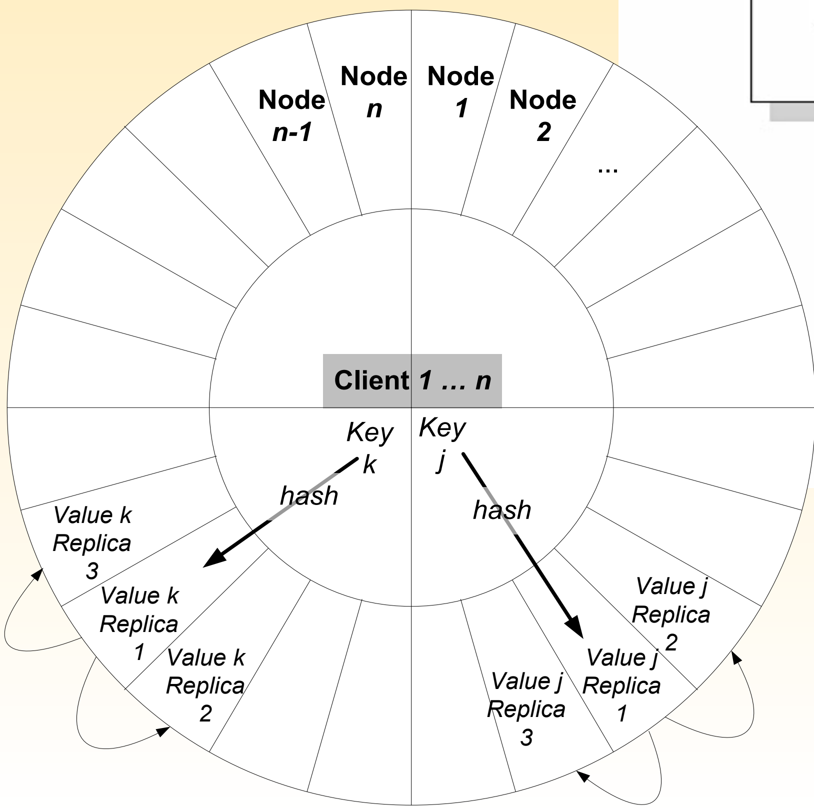
Big problem: file systems scalability

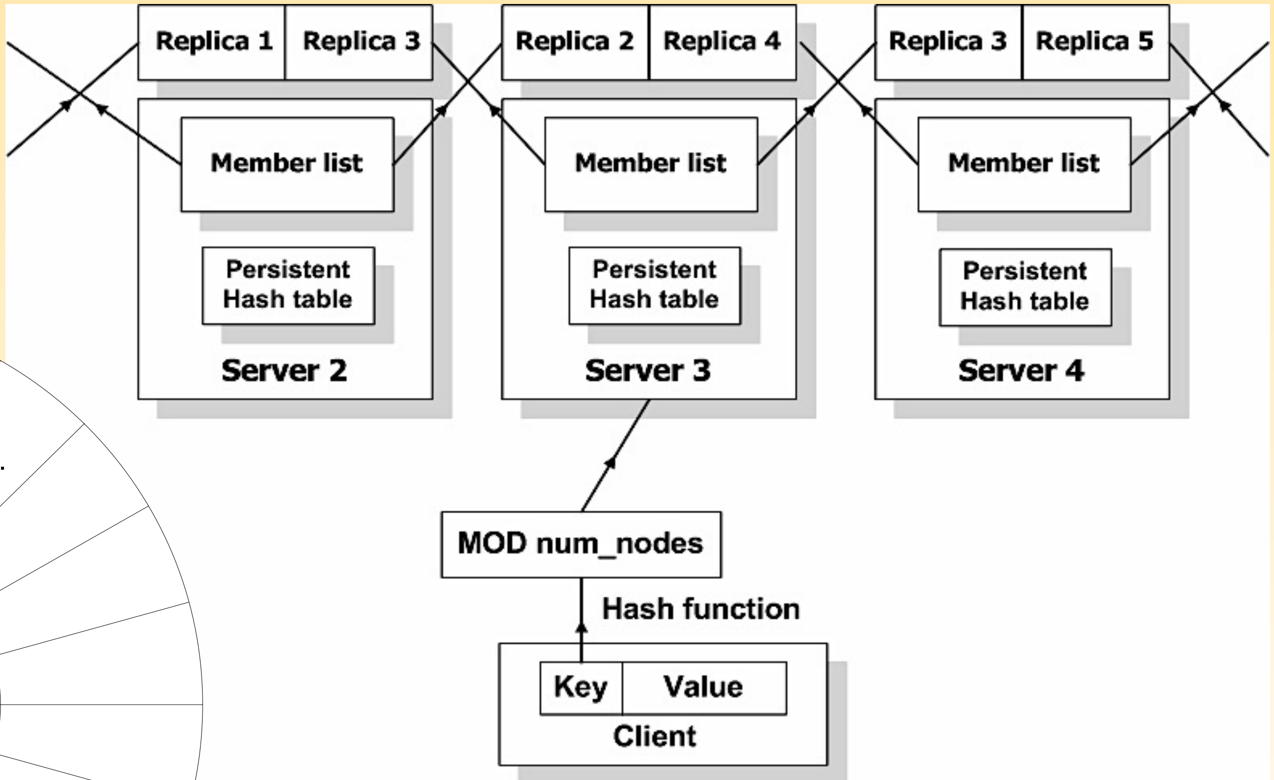
- Parallel file system (GPFS, PVFS, Lustre)
  - Separated computing resource from storage
  - Centralized metadata management
- Distributed file system(GFS, HDFS)
  - Specific-purposed design (MapReduce etc.)
  - Centralized metadata management

Proposed work

- A distributed hash table (DHT) for HEC
- As building block for high performance distributed systems
- Performance
  - Latency
  - Throughput
- Scalability
- Reliability

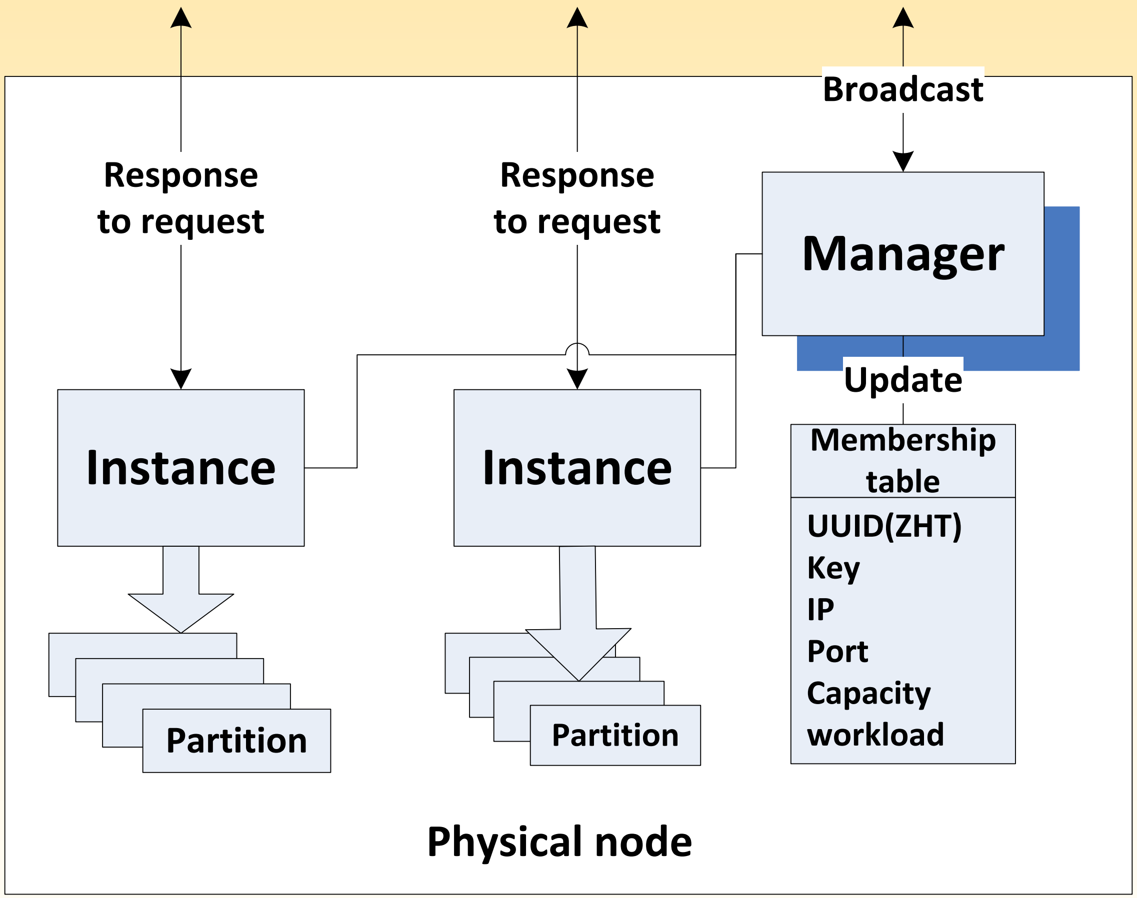
Zero-hop hash mapping





Architecture and terms

- Name space: 2^{64}
- Physical node
- Manager
- ZHT Instance
- Partition: n (fixed)
  - n = max(k)



Membership management

- Static: Memcached, ZHT
- Dynamic: Cassandra, Riak, ZHT
  - Logarithmic routing: $O(\log_k n)$ hops
    - 1-to-k networking
    - Small routing table: O(1): k
  - Constant routing: O(1) hops
    - Direct routing
    - All-to-all networking

Membership management in ZHT

- Update membership
  - Incremental broadcasting
- Remap k-v pairs
  - Traditional DHTs: rehash all influenced pairs
  - ZHT: Moving whole partition
    - HEC has fast local network!

Consistency

- Updating membership tables
  - Planed nodes join and leave: strong consistency
  - Nodes fail: eventual consistency
- Updating replicas
  - Configurable
  - Strong consistency: consistent, reliable
  - Eventual consistency: fast, availability

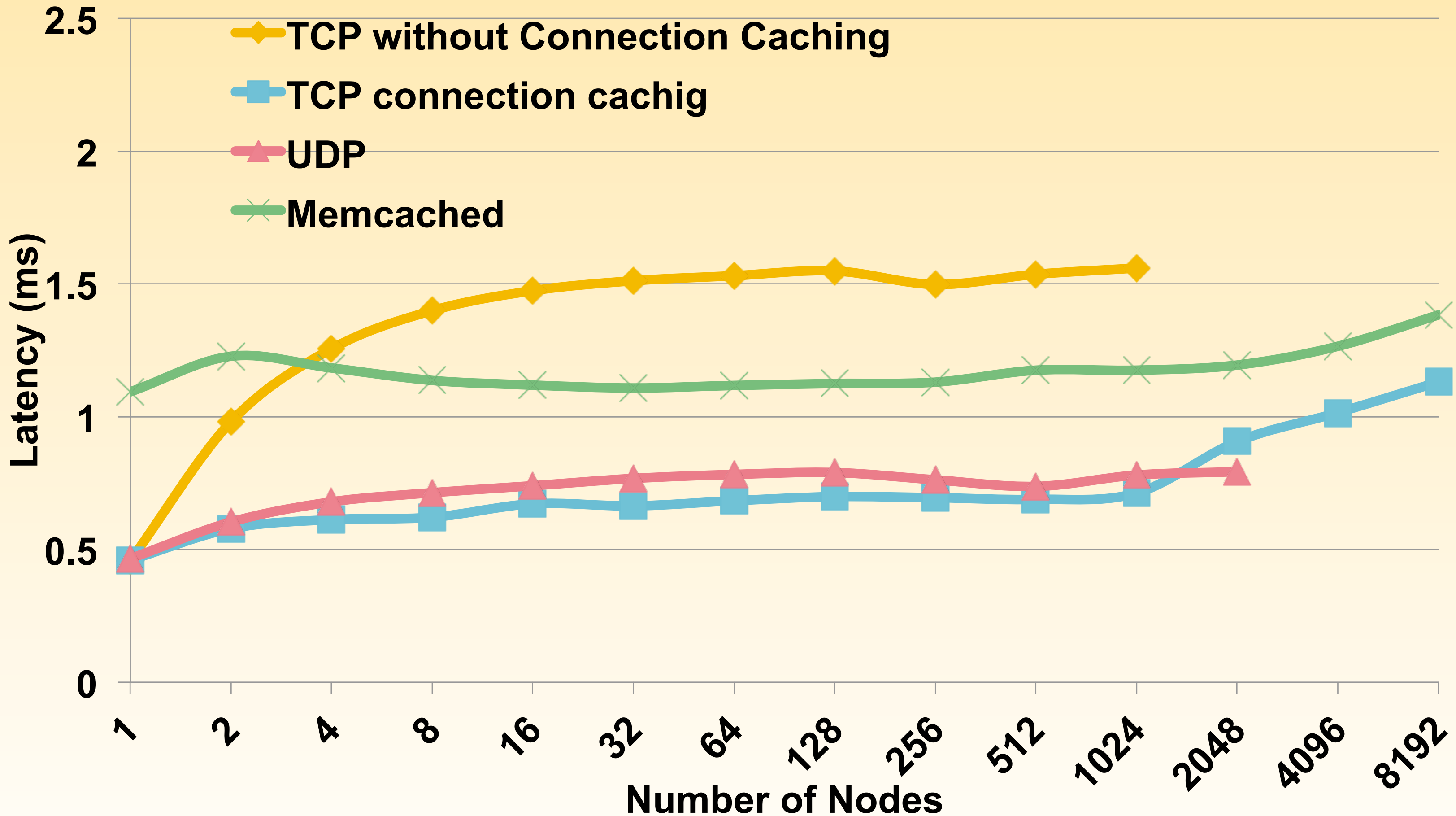
Failure handling

- Insert and append
  - Send it to next replica
  - Mark this record as primary copy
- Lookup
  - Get from next available replica
- Remove
  - Mark record on all replicas

Evaluation: test beds

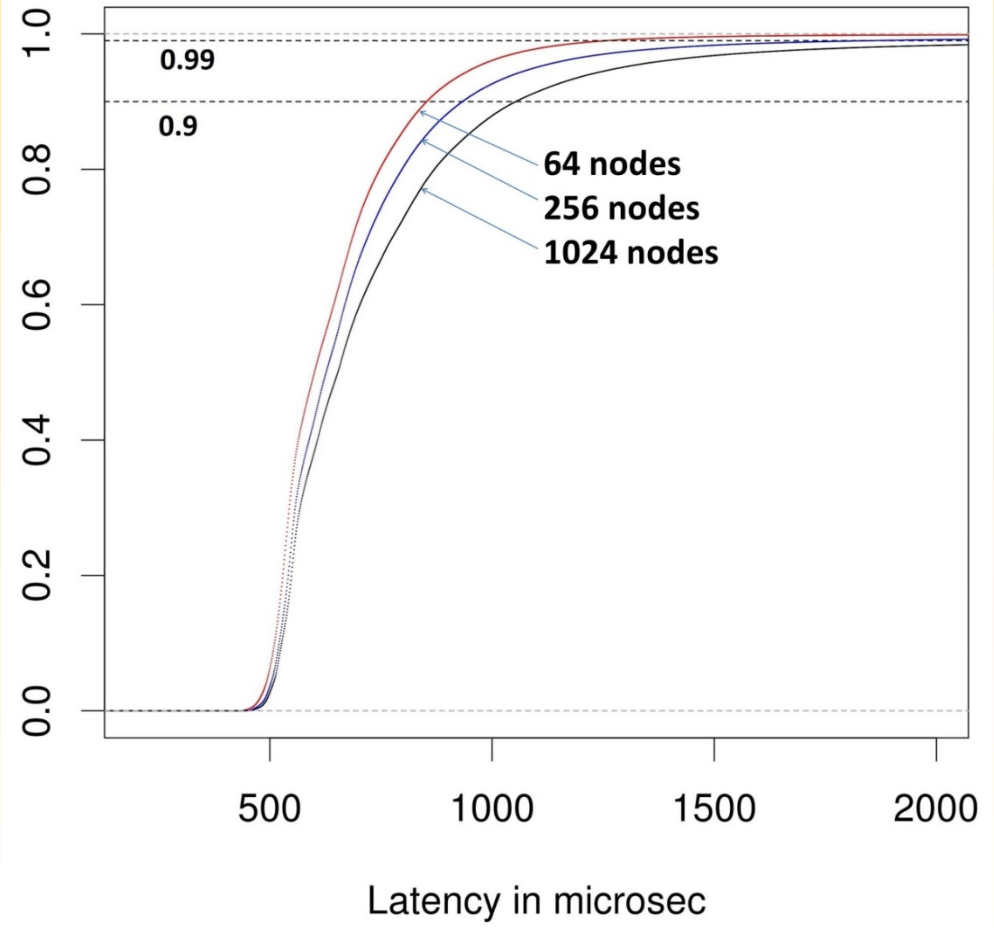
- IBM Blue Gene/P supercomputer
  - Up to 8192 nodes
  - 32768 instance deployed
- Commodity Cluster
  - Up to 64 node
- Amazon EC2
  - M1.medium and Cc2.8xlarge
  - 96 VMs, 768 ZHT instances deployed

Latency on BG/P



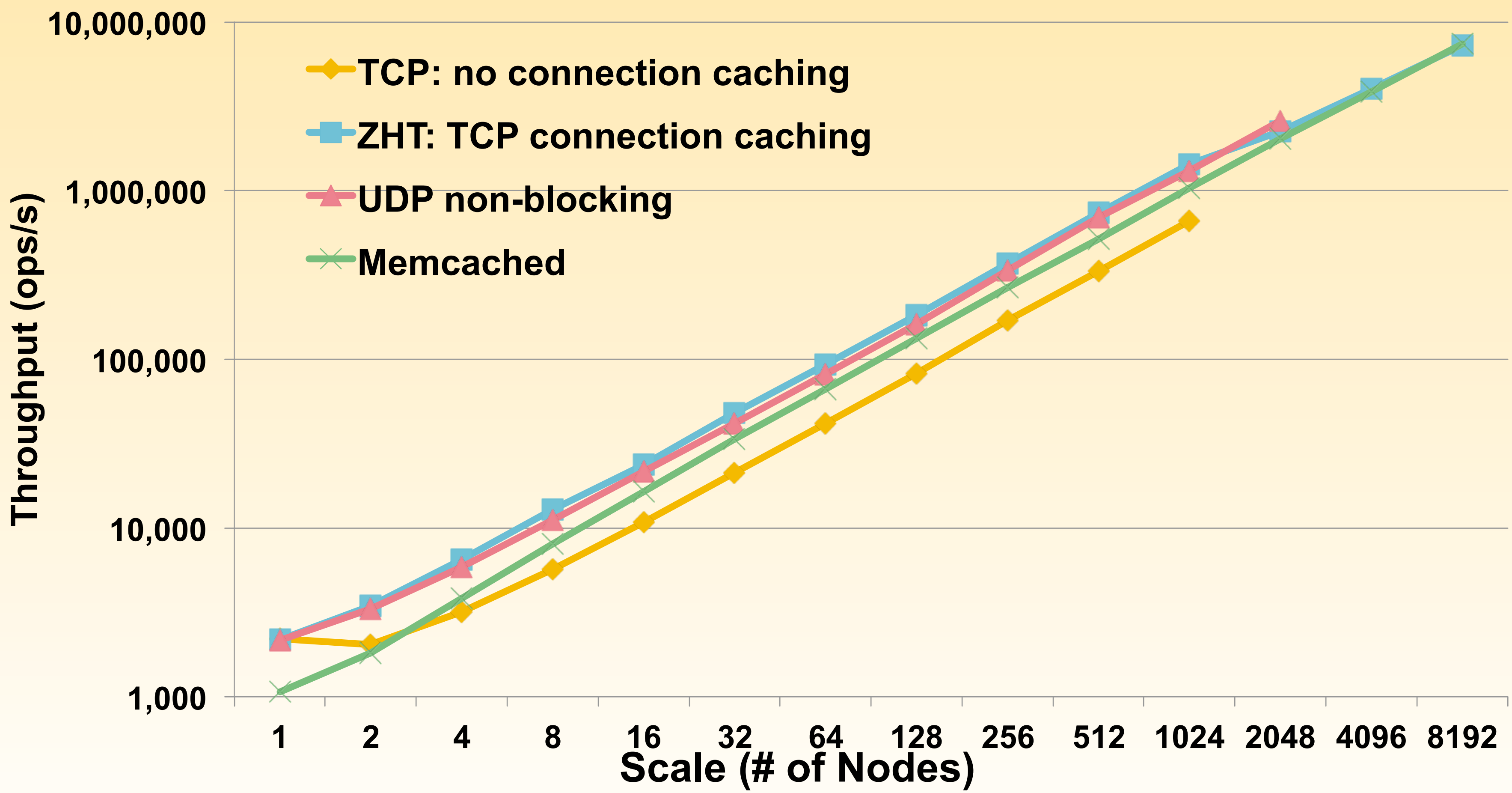
ILLINOIS INSTITUTE OF TECHNOLOGY

Latency distribution



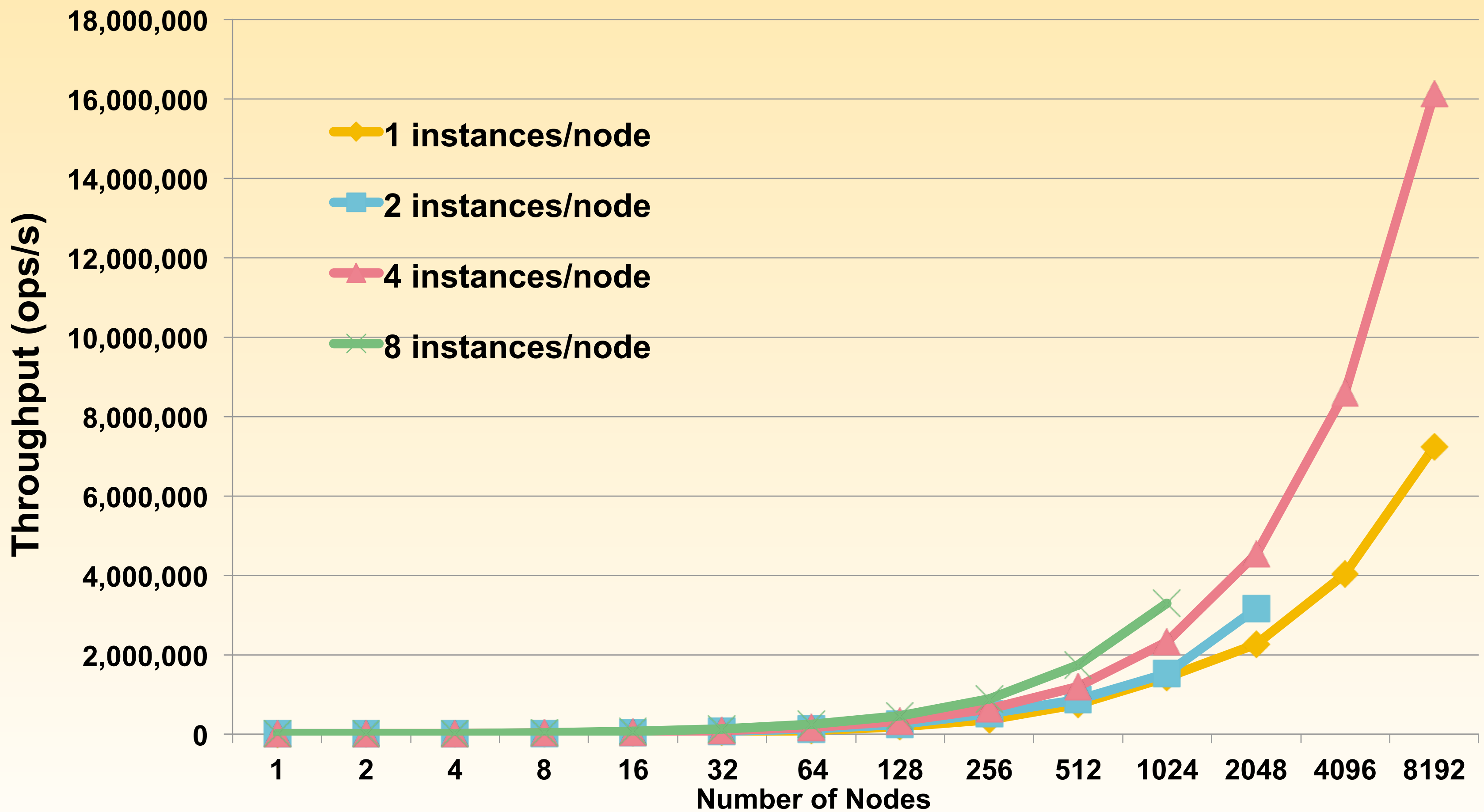
SCALES	75%	90%	95%	99%
64	713	853	961	1259
256	755	933	1097	1848
1024	820	1053	1289	3105

Throughput on BG/P



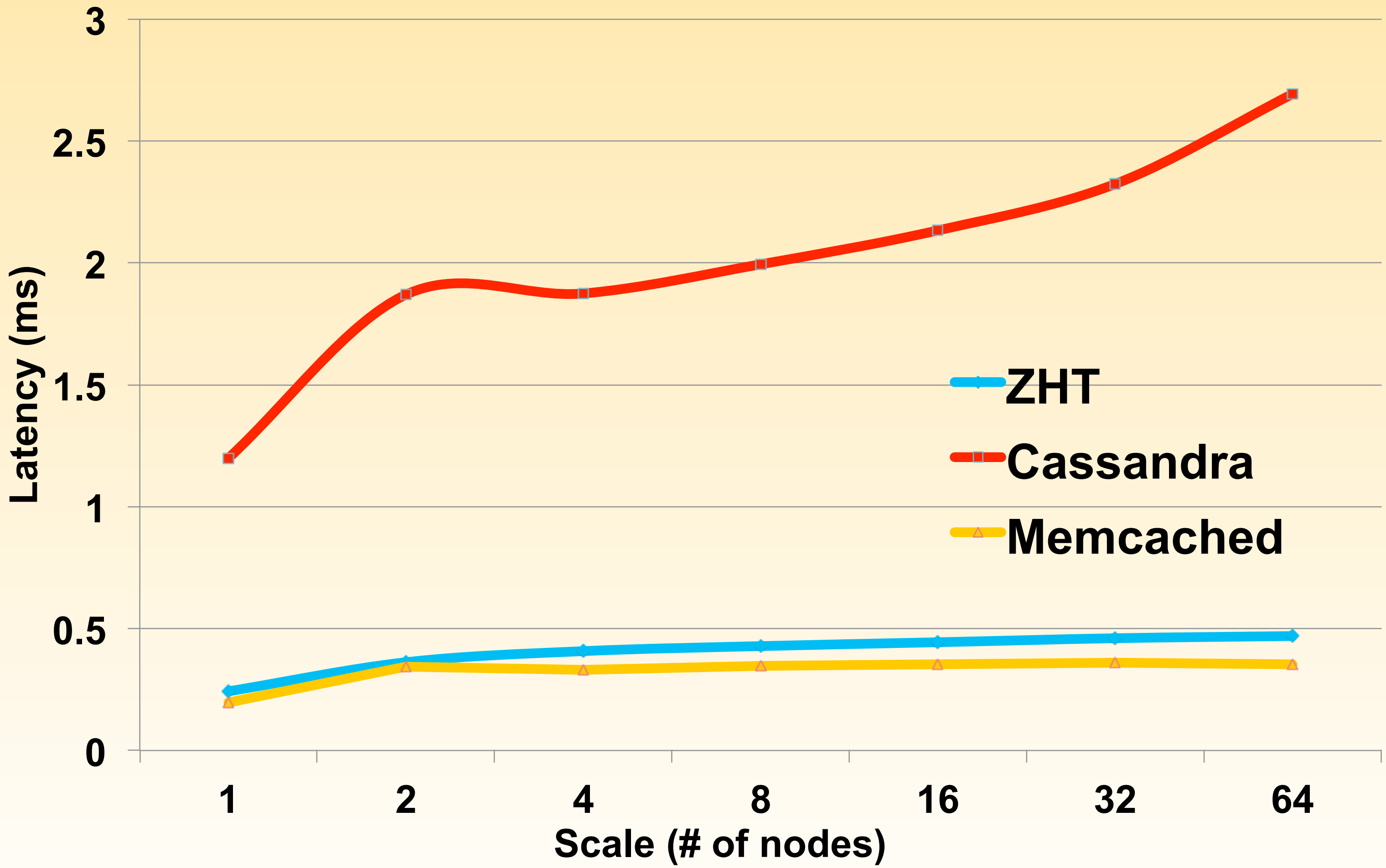
ILLINOIS INSTITUTE OF TECHNOLOGY

Aggregated throughput on BG/P



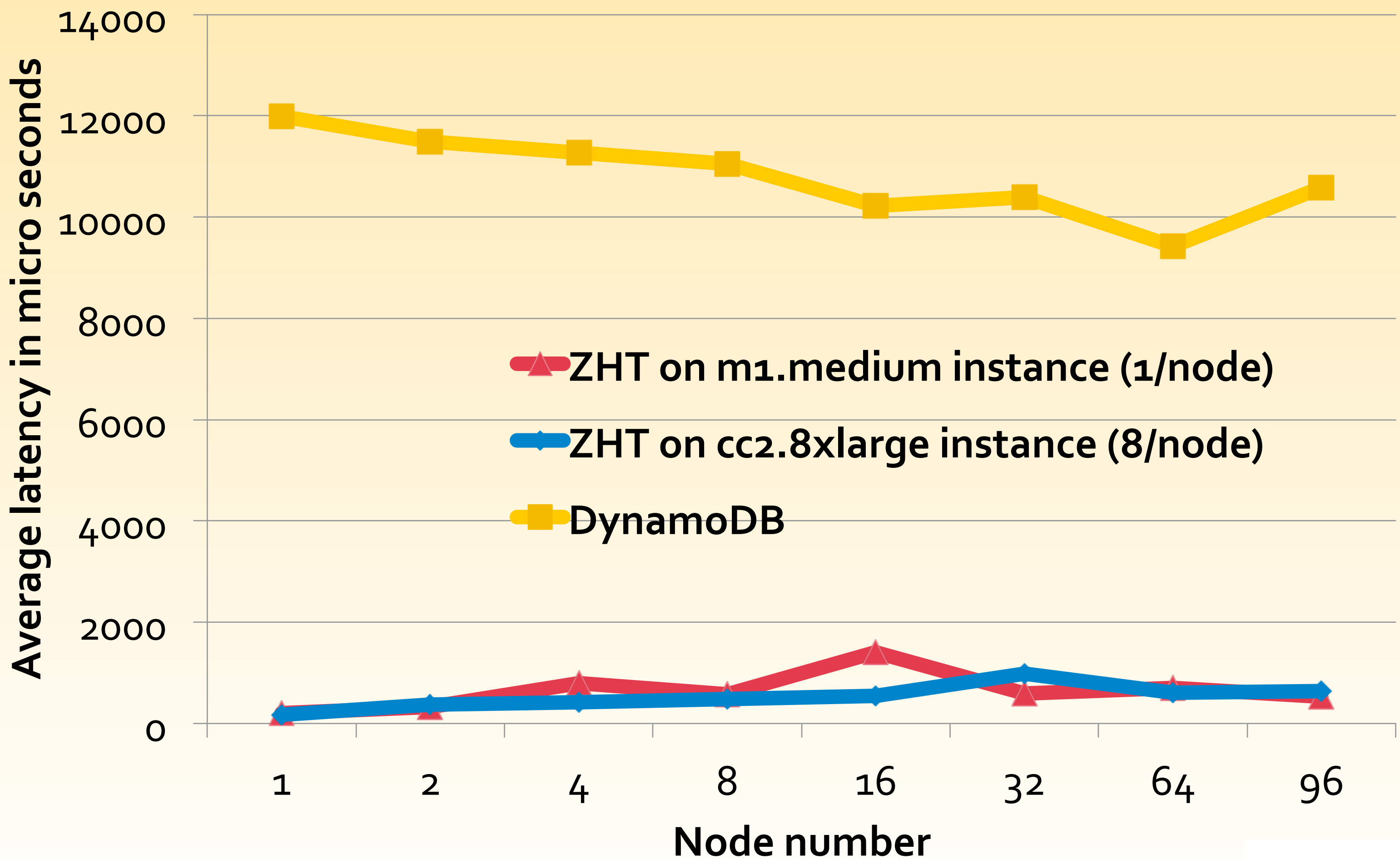
ILLINOIS INSTITUTE OF TECHNOLOGY

Latency on commodity cluster



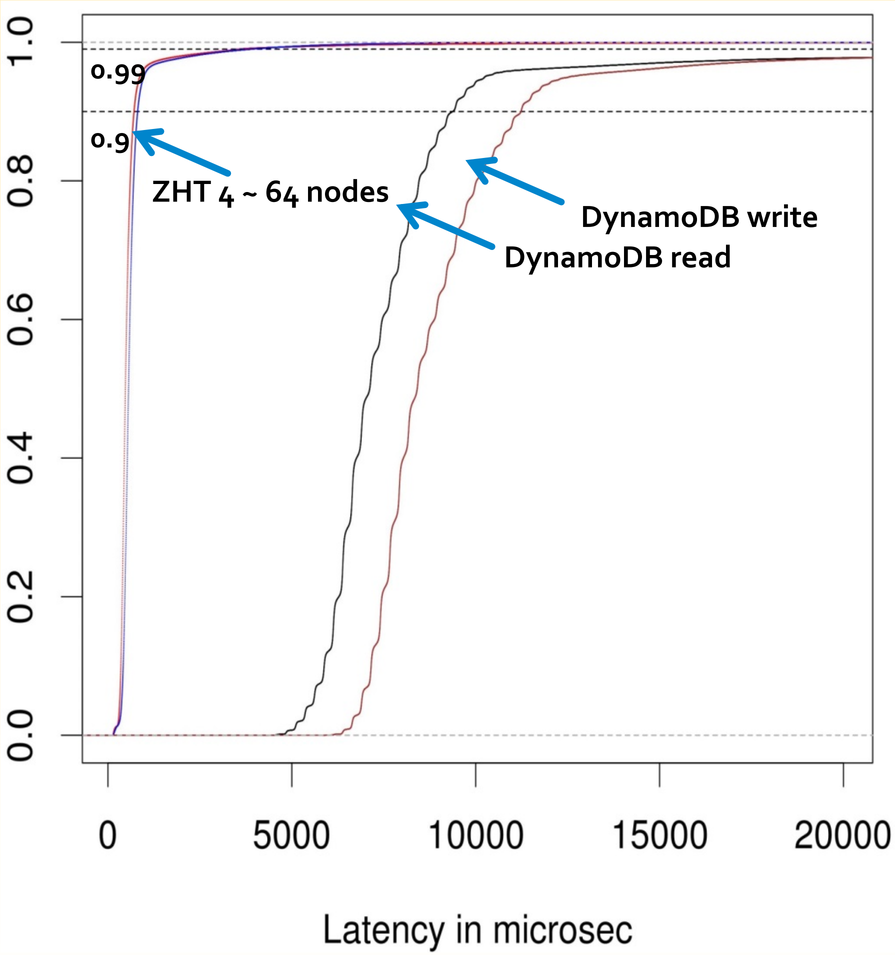
ILLINOIS INSTITUTE OF TECHNOLOGY

ZHT on cloud: latency



ILLINOIS INSTITUTE OF TECHNOLOGY

ZHT on cloud: latency distribution



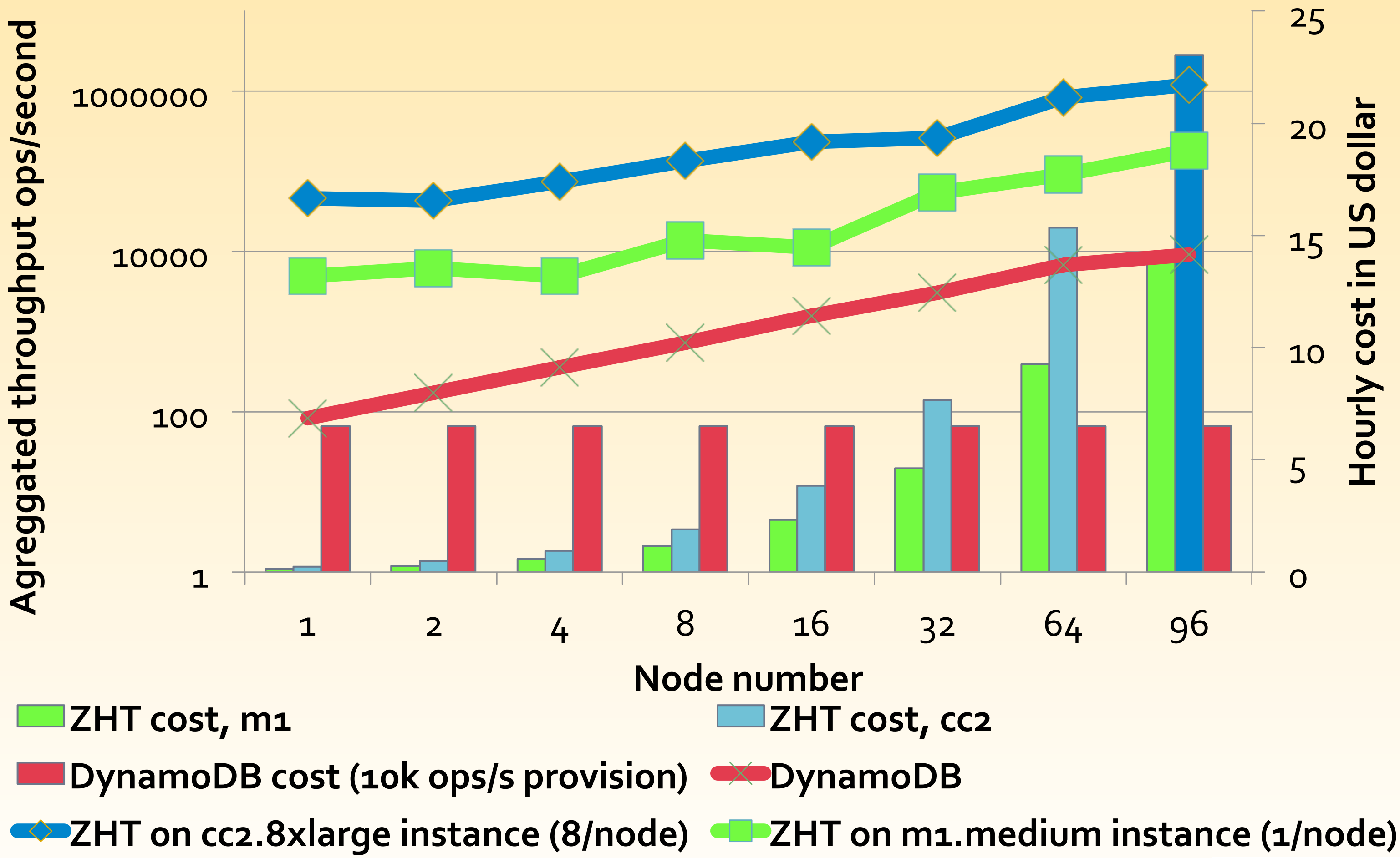
ZHT on cc2.8xlarge instance
8 s-c pair/instance

SCALES	75%	90%	95%	99%	AVG	THROUGHPUT
8	186	199	214	260	172	46421
32	509	603	681	1114	426	75080
128	588	717	844	2071	542	236065
512	574	708	865	3568	608	841040

DynamoDB: 8 clients/instance

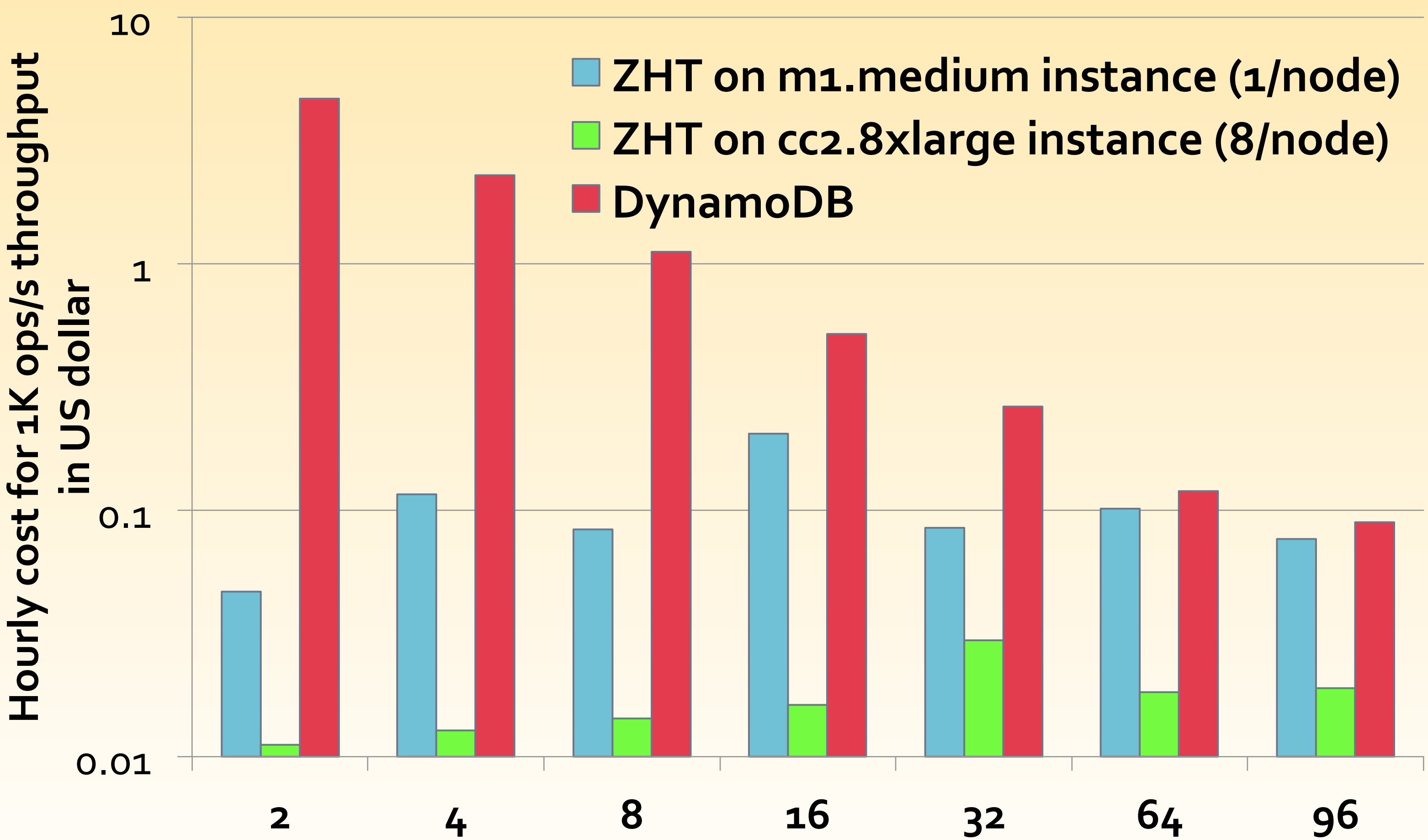
SCALES	75%	90%	95%	99%	AVG	THROUGHPUT
8	11942	13794	20491	35358	12169	83.39
32	10081	11324	12448	34173	9515	3363.11
128	10735	12128	16091	37009	11104	11527
512	9942	13664	30960	38077	28488	ERROR

ZHT on cloud: throughput



ILLINOIS INSTITUTE OF TECHNOLOGY

Amortized cost



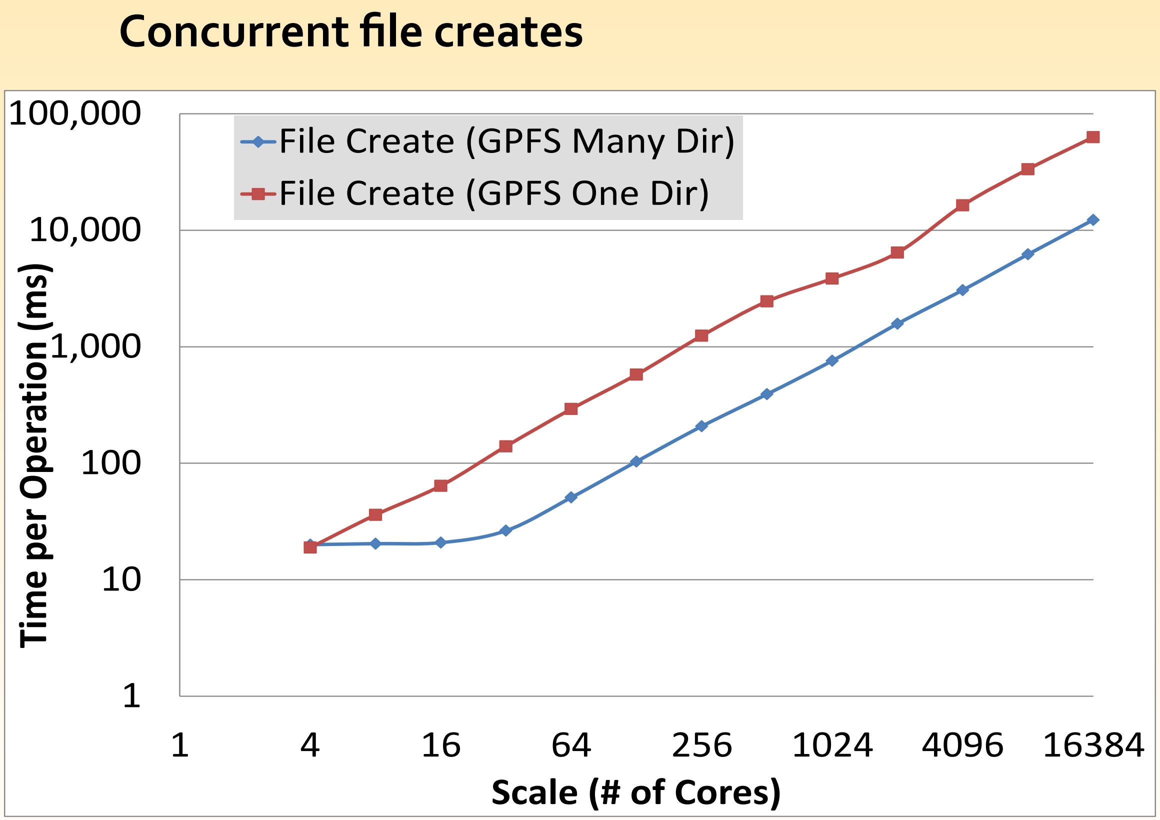
ILLINOIS INSTITUTE OF TECHNOLOGY

Applications

- FusionFS
  - A distributed file system
  - Metadata: ZHT
- IStore
  - A information dispersal storage system
  - Metadata: ZHT
- MATRIX
  - A distributed many-Task computing execution framework
  - ZHT is used to submit tasks and monitor the task execution status

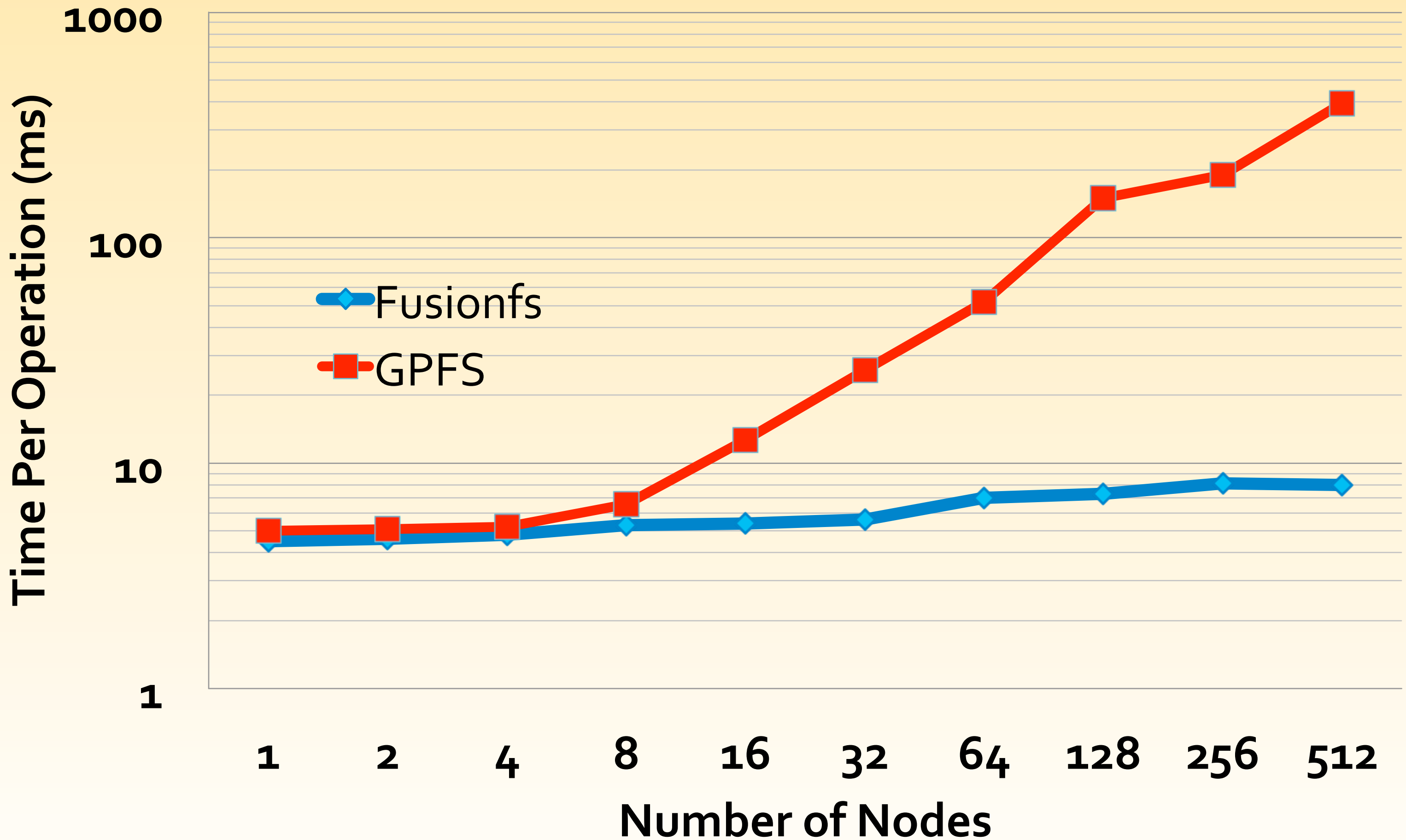
The bottleneck of file systems

- **Metadata**



ILLINOIS INSTITUTE OF TECHNOLOGY

FusionFS result: Concurrent File Creates



ILLINOIS INSTITUTE OF TECHNOLOGY

Related work: Distributed Hash Tables

- Many DHTs: Chord, Kademlia, Pastry, Cassandra, C-MPI, Memcached, Dynamo ...
- Why another?

Name	Impl.	Routing Time	Persistence	Dynamic membership	Append Operation
Cassandra	Java	Log(N)	Yes	Yes	No
C-MPI	C	Log(N)	No	No	No
Dynamo	Java	0 to Log(N)	Yes	Yes	No
Memcached	C	0	No	No	No
ZHT	C++	0 to 2	Yes	Yes	Yes

ILLINOIS INSTITUTE OF TECHNOLOGY

Conclusion

- ZHT : A distributed Key-Value store
  - light-weighted
  - high performance
  - Scalable
  - Dynamic
  - Fault tolerant
  - Versatile: works from clusters, to clouds, to supercomputers

Questions?

Tonglin Li
tli13@hawk.iit.edu
http://datasys.cs.iit.edu/projects/ZHT/

ILLINOIS INSTITUTE OF TECHNOLOGY