ZHT

A Zero-hop distributed Hash Table for high-end computing systems

Tonglin Li

Acknowledgements

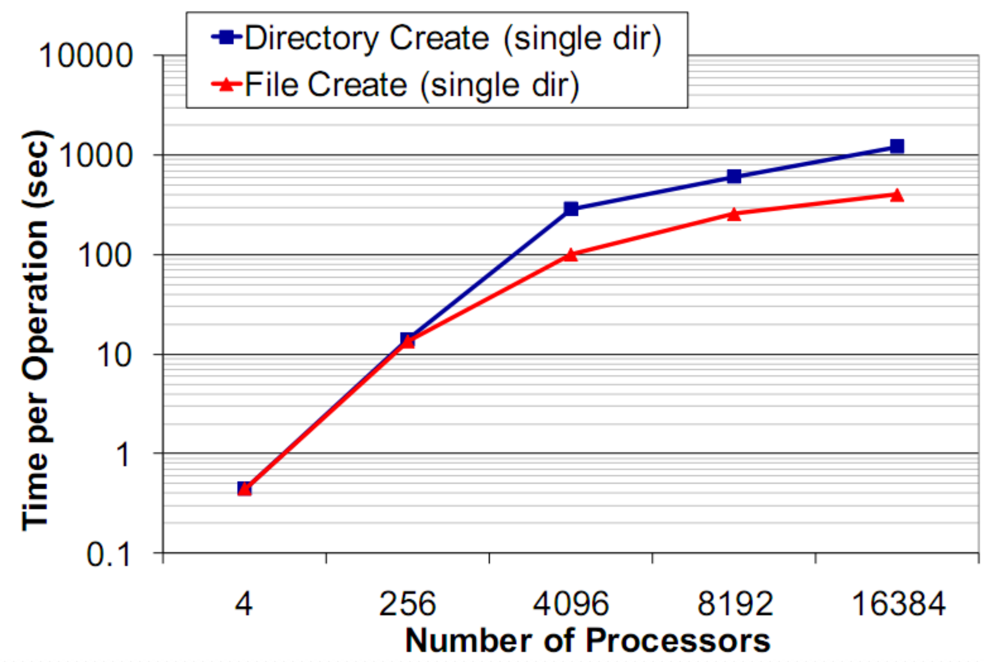
- I'd like to thank Dr. Ioan Raicu for his support and advising, and the help from Raman Verma, Xi Duan, and Hui Jin.
- This work is published in HPDC/SigMetrics 2011 poster session.

Background

- Data: files
- **Metadata:** data about files
- Distributed Storage System

State-of-art metadata management

- Relational DB based metadata
  - Heavy
- Centralized metadata management
  - Communication jam
  - Fragile
  - Not scalable
- Typical parallel file System: GPFS by IBM



Proposed work: a new DHT for metadata management

- What is a DHT?
- Why DHT?
  - Fully distributed: no centralized bottleneck
  - High performance: high aggregated I/O
  - Fault tolerance
- But existing DHTs are not fast enough.
  - Slow and heavy
  - High latency

Related work: DHT

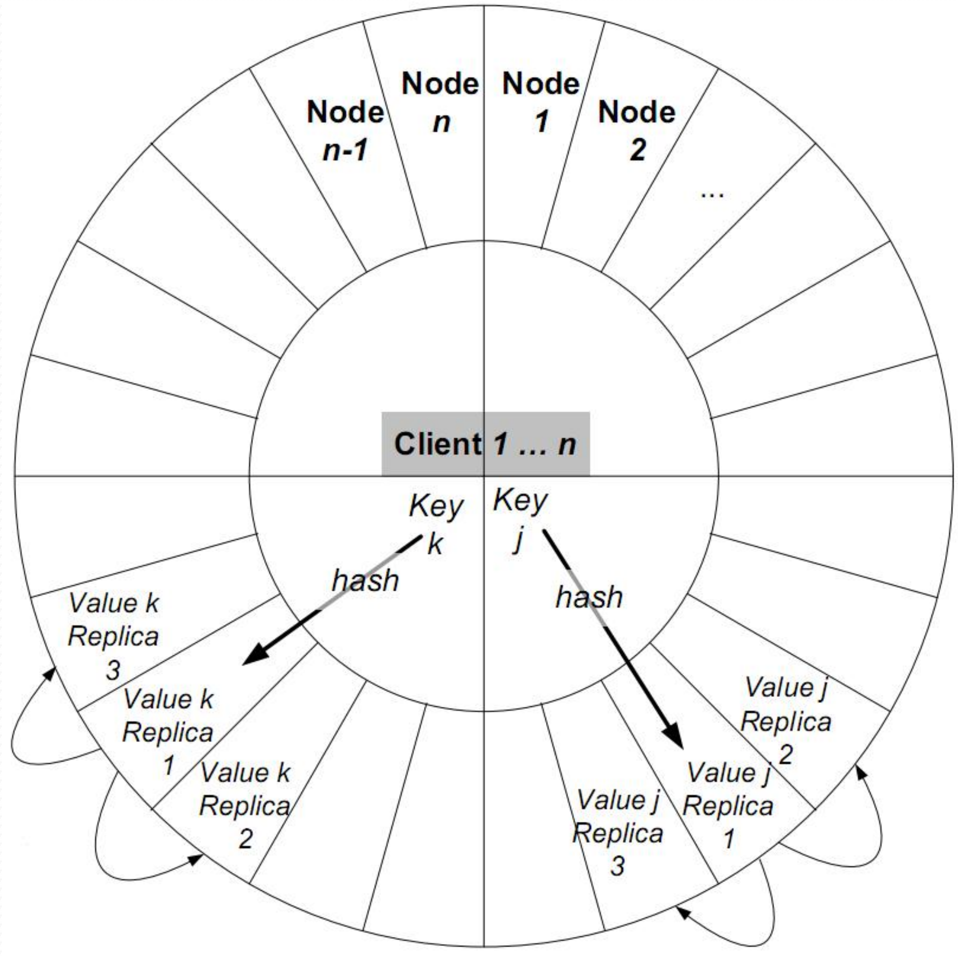
	Architecture Topology	Routing Time(hops)
Chord	Ring	Log(N)
CAN	Virtual multidimensional Cartesian coordinate space on a multi-torus	$O(dn^{l/d})$
Pastry	Hypercube	O(logN)
Tapestry	Hypercube	$O(\log_B N)$
Cycloid	Cube-connected-cycle graph	O(d)
Kademlia	Ring	Log(N)
Memcached	Ring	2
C-MPI	Ring	Log(N)
Dynamo	Ring	0

Practical assumptions of HEC

- Reliable hardware
- Fast network interconnects
- Non-existent node “churn”
- Batch oriented: steady amount of resource

Overview of Design

- Zero-hop



Implementation: Persistency

- Database or key-value store
  - Relational database: transaction, complex query
    - BerkeleyDB, MySQL
  - Key-value store: small, simple, fast, flexible
    - Kyotocabinet, CouchDB, HBase
- Log recording and playback
  - Bootstrap system requires to playback all log records for loading metadata

Implementation: Failure handling

- Insert
  - If one try failed: send it to closest replica
  - Mark this record as primary copy
  - Recover to original node when reboot system
- Lookup
  - If one try fail: try next one, until go through all replicas
- Remove
  - Mark record removed(but not really remove)
  - Recover to original node when reboot system

Membership management

- Static member list
  - reliable hardware
  - non-existent node “churn”
- If a node quit, it never come back
- Consistent hashing
  - Remove a node doesn't impact the hash map much

Replication update

- Server-side replication
- Asynchronized update
- Sequential update among replicas
  - P->R1; R1->R2; R2->R3

Performance evaluation

- Hardware: SiCortex SC5832
  - 970 nodes
  - 4GB RAM/node
  - 5,832 cores
- OS: Cento OS 5.0 (Linux)
- Batch execution system: SLURM

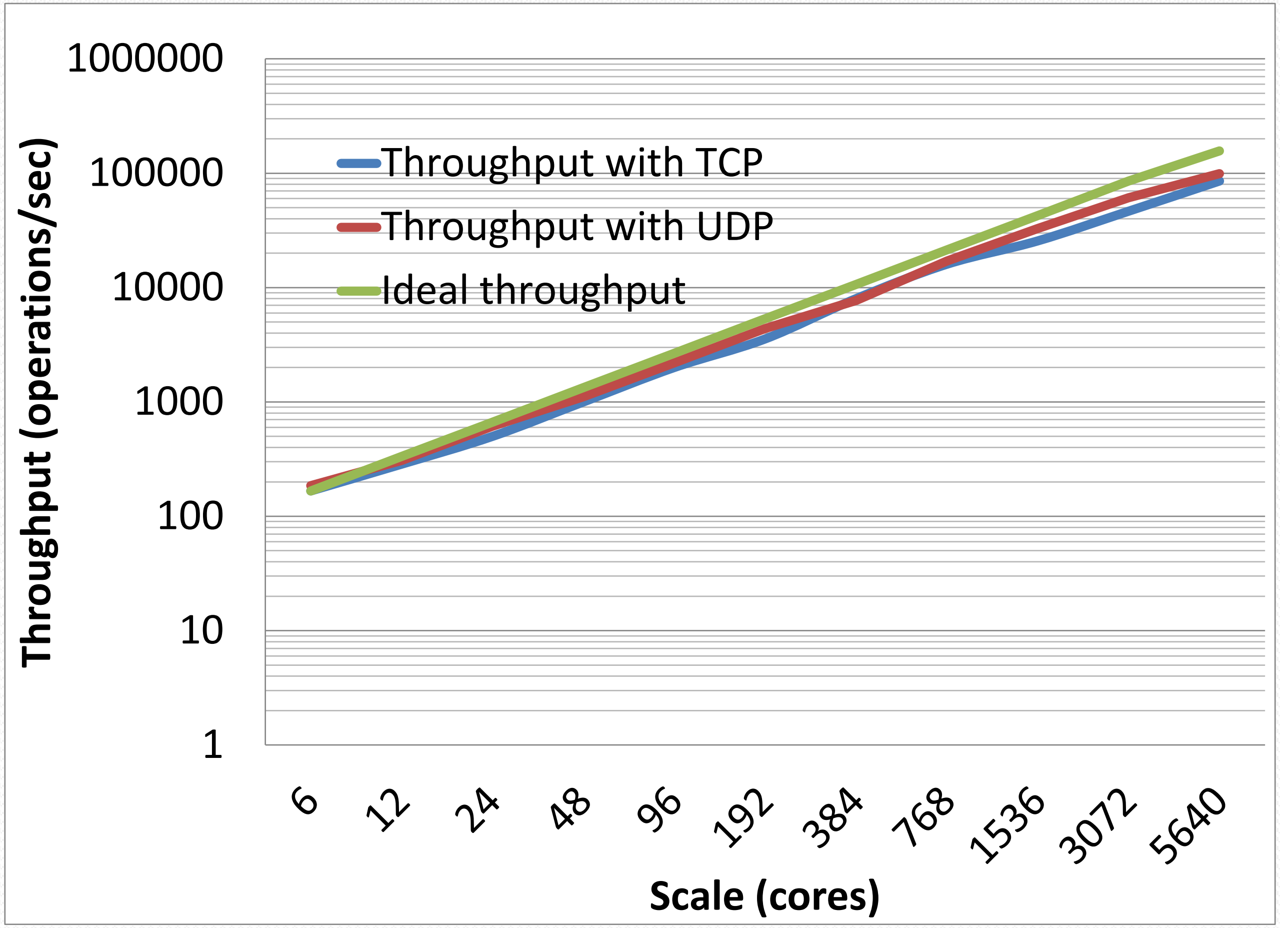
Throughput

- Ideal throughput:

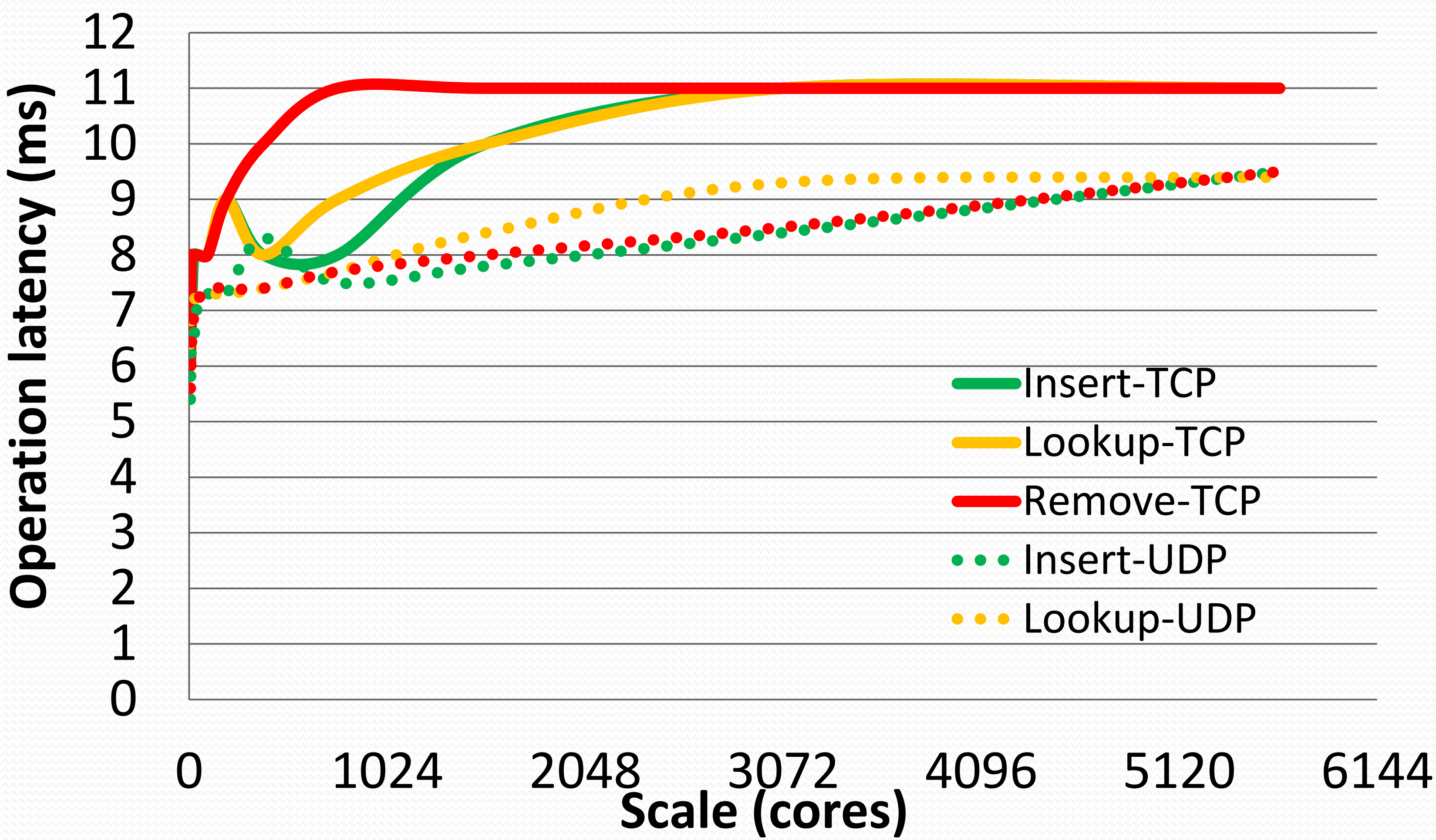
  T_i = tested single node throughput * node number
- Measured throughput :

  T_a= Sum of all single node tested throughputs

Ideal vs. measured throughput

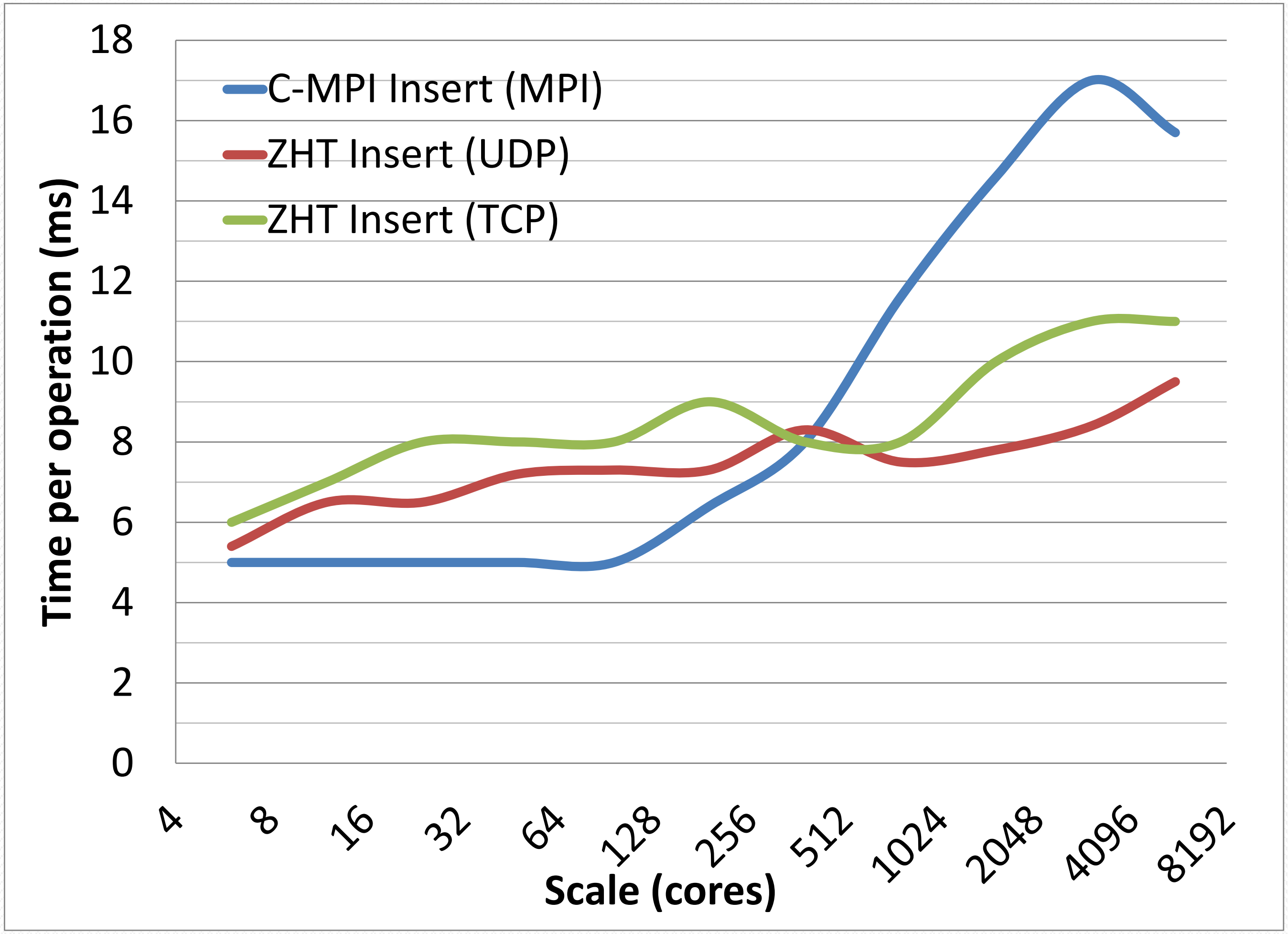


TCP v.s. UDP



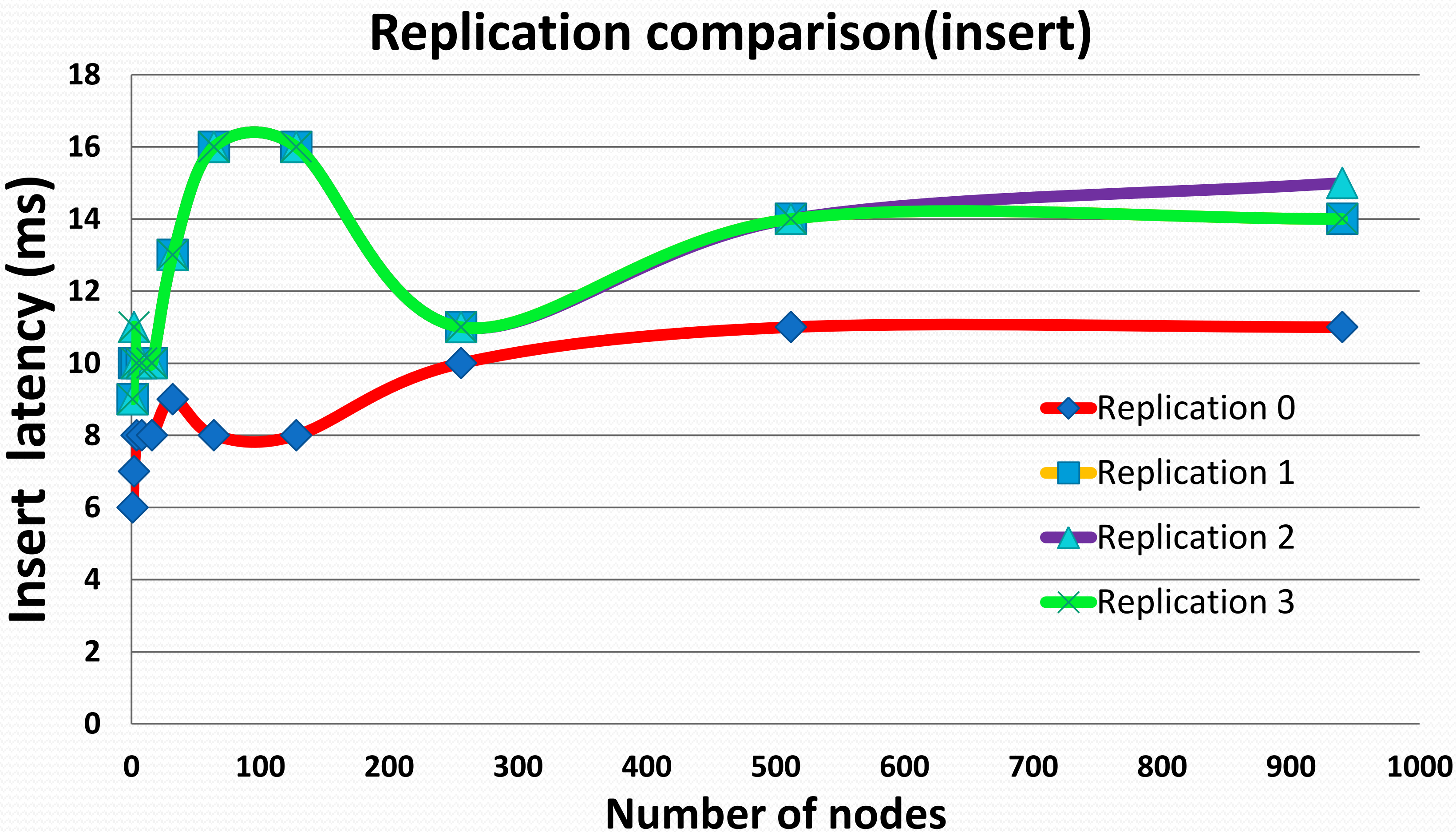
Operation latency (ms)
12
11
10
9
8
7
6
5
4
3
2
1
0
0
1024
2048
3072
4096
5120
6144
Scale (cores)
Insert-TCP
Lookup-TCP
Remove-TCP
Insert-UDP
Lookup-UDP

ZHT v.s. C-MPI

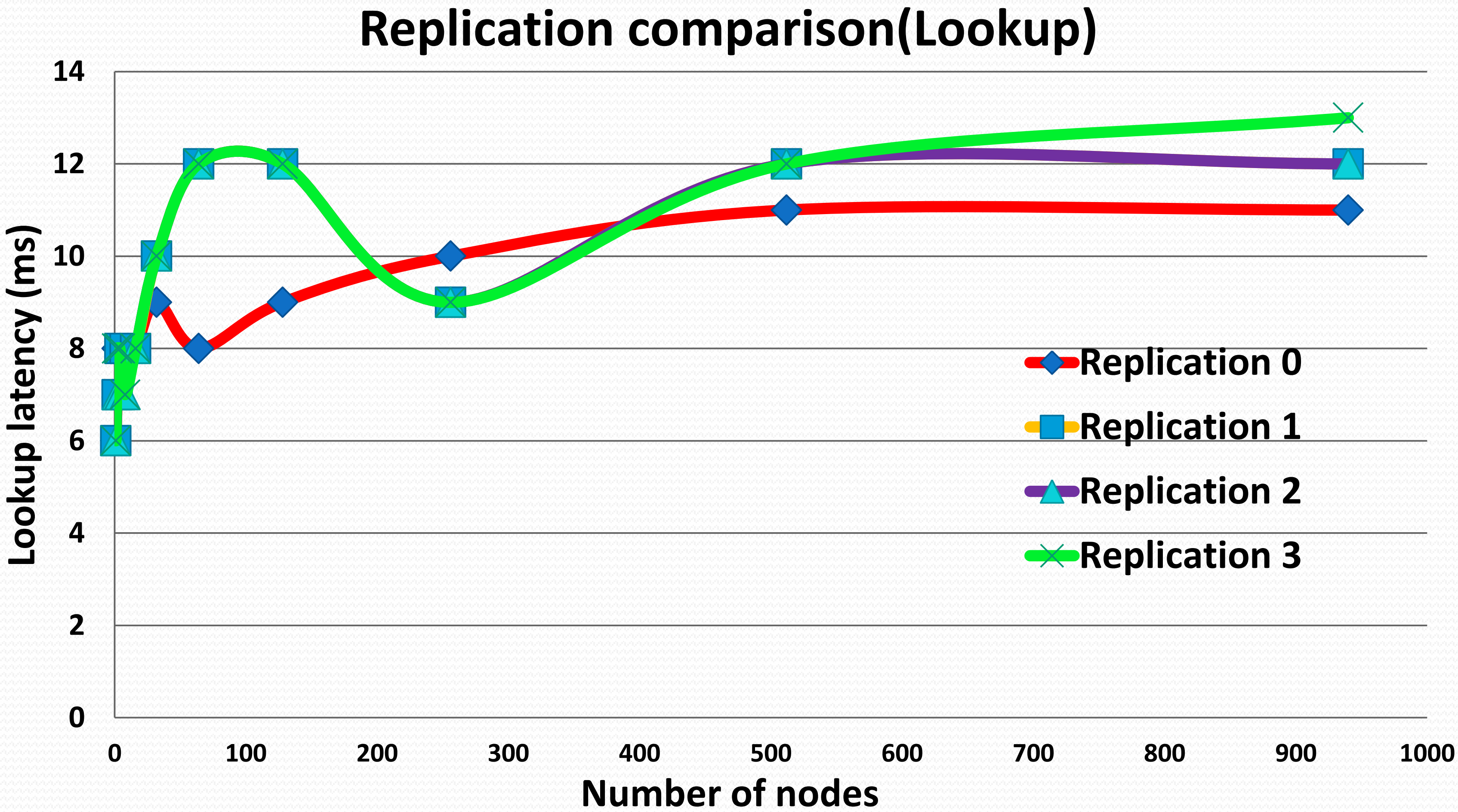


C-MPI Insert (MPI)
ZHT Insert (UDP)
ZHT Insert (TCP)
Time per operation (ms)
0
2
4
6
8
10
12
14
16
18
4
8
16
32
64
128
256
512
1024
2048
4096
8192
Scale (cores)

Replication overhead



Replication overhead



Future work

- Comprehensive fault tolerance
- Dynamic membership management
- More protocol support (MBI…)
- Merge with FusionFS
- Data aware job scheduling
- Many optimizations
- Larger scale evaluation (BlueGene/P, etc)

Conclusion

ZHT offer a good solution of distributed key-value store, they are

- Light-weighted: cost less than 10MB memory/node
- Scalable: near-linearly scales up to 5000 cores
- Very fast: 100,000 operations/ sec
- Low latency: about 10ms
- Wide range of use: open source

Questions?