MATRIX: MAny-Task computing execution fabRIc for eXtreme scales

ILLINOIS INSTITUTE OF TECHNOLOGY

Anupam Rajendran
Department of Computer Science
Illinois Institute of Technology
arajend5@hawk.iit.edu

Ke Wang
Department of Computer Science
Illinois Institute of Technology
kwang22@hawk.iit.edu

Ioan Raicu
Department of Computer Science
Illinois Institute of Technology
iraicu@cs.iit.edu

DataSys Data-Intensive Distributed Systems Laboratory

Motivation and Goal

- State-of-the-art job schedulers have master/slave architecture, where the master becomes a performance bottleneck and is susceptible to single point of failure, especially at petascales
- Develop a dynamic distributed scalable job scheduling system at the granularity of node/core levels with distributed load balancing algorithm (work stealing) leading to high throughput and system utilization.

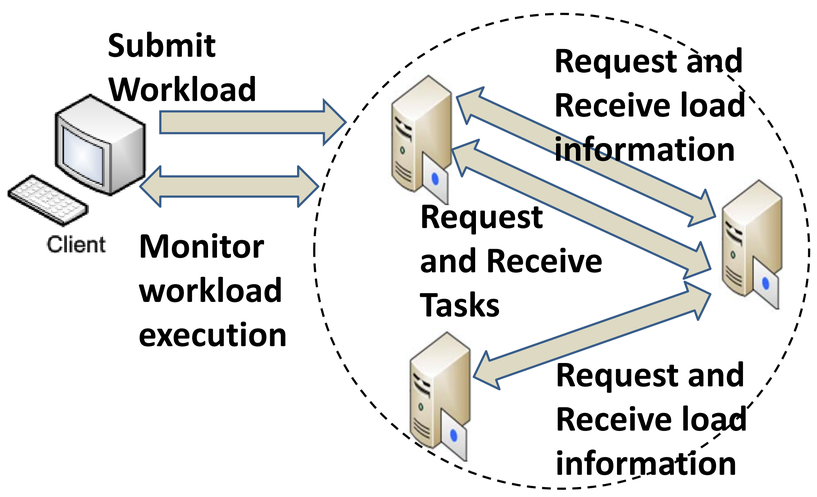
Building Blocks

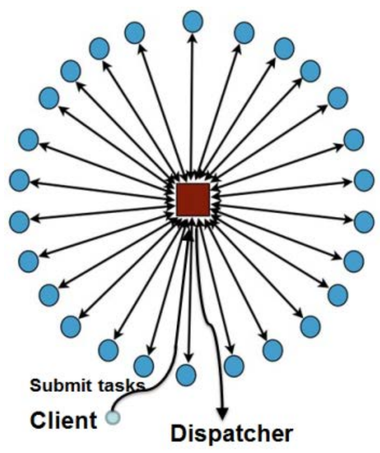
- **MATRIX**: a task execution framework which applies work stealing algorithm for achieving efficient distributed job scheduling
  - Support single core tasks, single node tasks or multi-node tasks
  - Also supports task dependency
  - Scales up to 4096 cores
- **ZHT**: a distributed key-value store designed for extreme-scales
  - Distributed metadata and data management
  - Scales up to 32K-cores

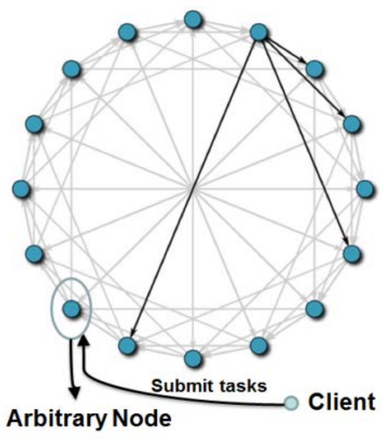
MATRIX Architecture

➢ ***Workload Submission***

- The task dispatcher can submit the entire workload either to one compute node (worst case) or spread it evenly across multiple nodes (best case)
- For the worst case the work stealing algorithm quickly distributes the load evenly across all the compute nodes to get full system utilization and high throughput







➢ ***Workload Execution***

1. Client generates a workload of tasks, submits the workload to the compute nodes and periodically monitors the status
2. The compute nodes execute the tasks in the workload
3. Idle nodes gather load information from busy nodes
4. Then they find the node with heaviest load and steal tasks from that node

Performance on BlueGene/P

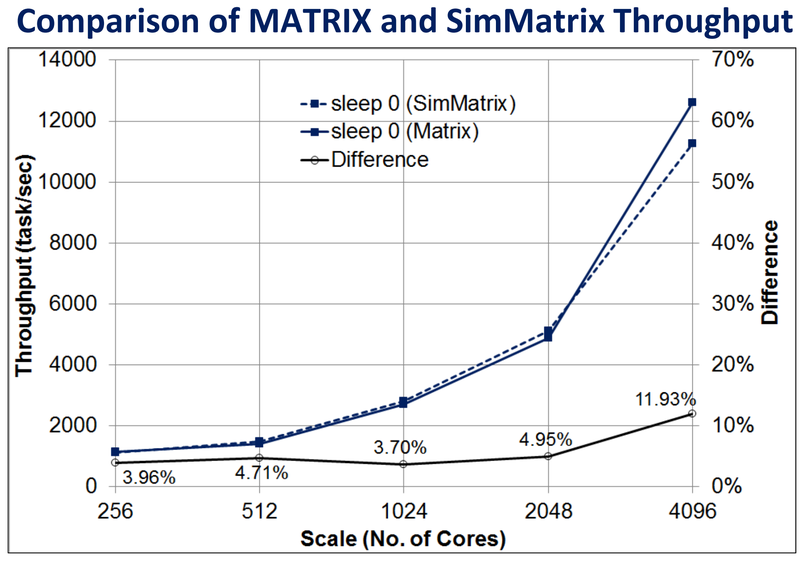
Parameter Space

- Task duration – 64ms ~ 8s
- Type of workload– Bag of tasks (no task dependency), Fan-In DAG, Fan-Out DAG, Pipeline DAG
- Number of nodes – 64 ~ 1024 nodes

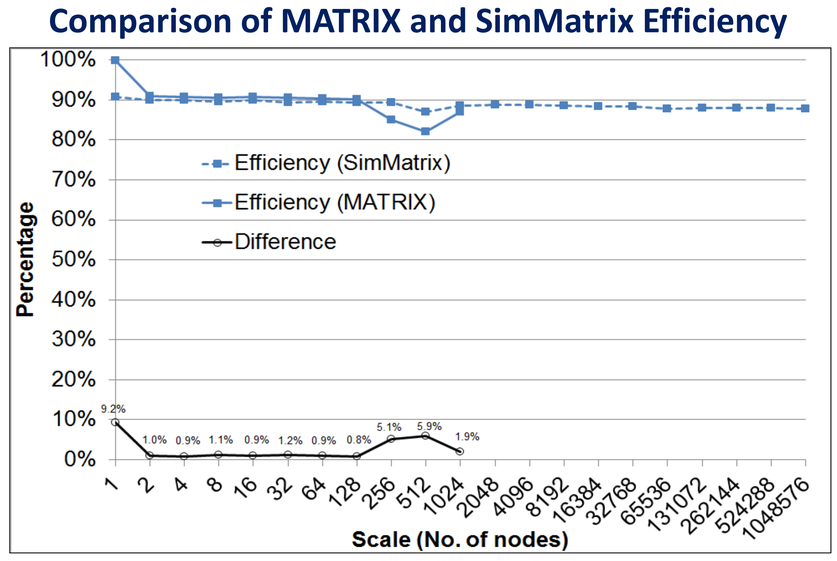
Metrics

- Throughput
- Efficiency
- System Utilization
- Network Traffic

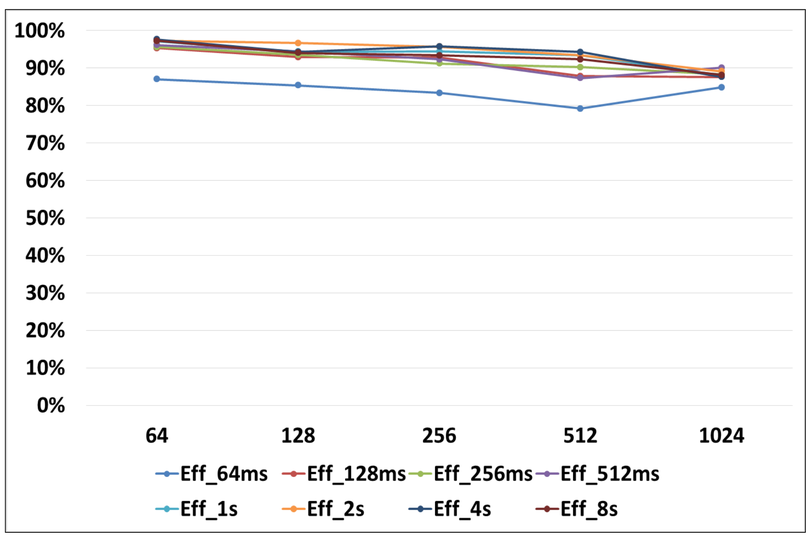
Comparison of MATRIX and SimMatrix Throughput



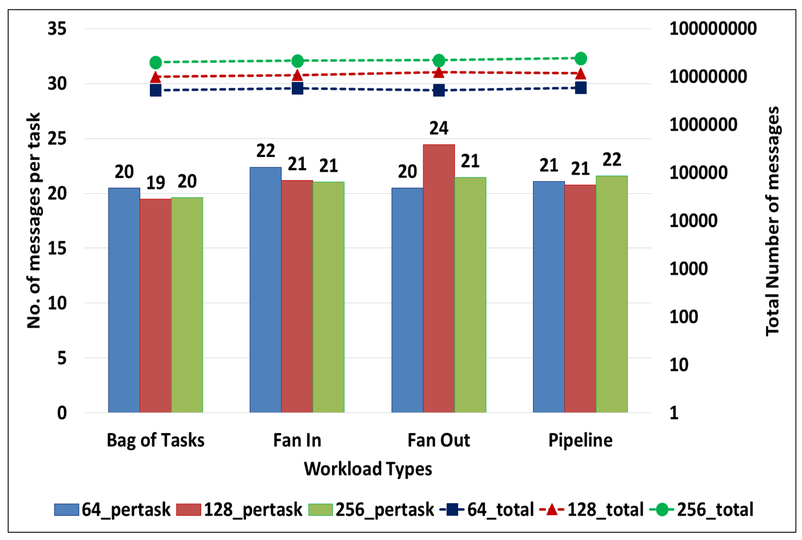
Comparison of MATRIX and SimMatrix Efficiency



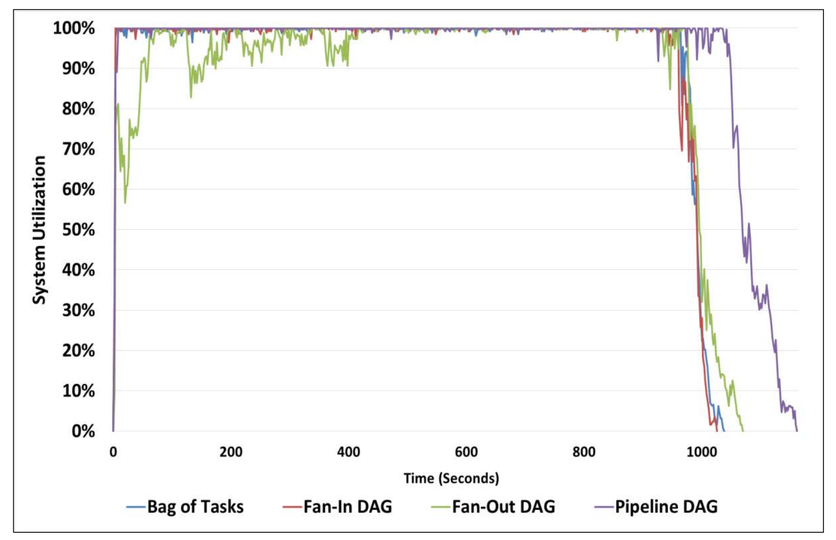
Scalability of Adaptive Work Stealing algorithm for tasks of different granularities



No. of messages per task for different workloads



System Utilization for different types of workload



Results and Conclusion

- Real implementation matches the simulation results with 4% difference
- Excellent scalability with about constant efficiency (80%+) for workload of different granularities up to 4096 cores
- System utilization is 100% at most of the time thus achieving excellent throughput
- Number of messages per-task for different types of workload is about constant

References

[1] I. Raicu, Y. Zhao, I. Foster, "Many-Task Computing for Grids and Supercomputers," 1st IEEE Workshop on Many-Task Computing on Grids and Supercomputers (MTAGS) 2008.

[2] J. Dinan, D.K. Larkins, P. Sadayappan, S. Krishnamoorthy, J. Nieplocha, "Scalable work stealing," In Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis, 2009.

[3] Tonglin Li, Xiaobing Zhou, Kevin Brandstatter, Dongfang Zhao, Ke~Wang, Anupam Rajendran, Zhao Zhang, and Ioan Raicu. ZHT: A Light-weight Reliable Persistent Dynamic Scalable Zero-hop Distributed Hash Table. IEEE IPDPS, Boston, MA, 2013, to appear.

Future Work

- Integrate with Slurm job manager for HPC workloads
- Integrate with Swift for running real scientific application
- Add network topologies such as logarithmic topology to allow each compute node select neighbors based on location to optimize network traffic
- Develop and improve scalability of MATRIX

Acknowledgement

This work is supported by NSF grant OCI-1054974.
Special thanks to Tonglin Li, Xiaobing Zhou, Kevin Brandstatter and Zhao Zhang for their collaboration.