Exploring Distributed HPC Scheduling with Randomized Resource Stealing

Kiran Ramamurthy
Dept. of Computer Science
Illinois Institute of Technology

Dr. Ioan Raicu
Dept. of Computer Science
Illinois Institute of Technology

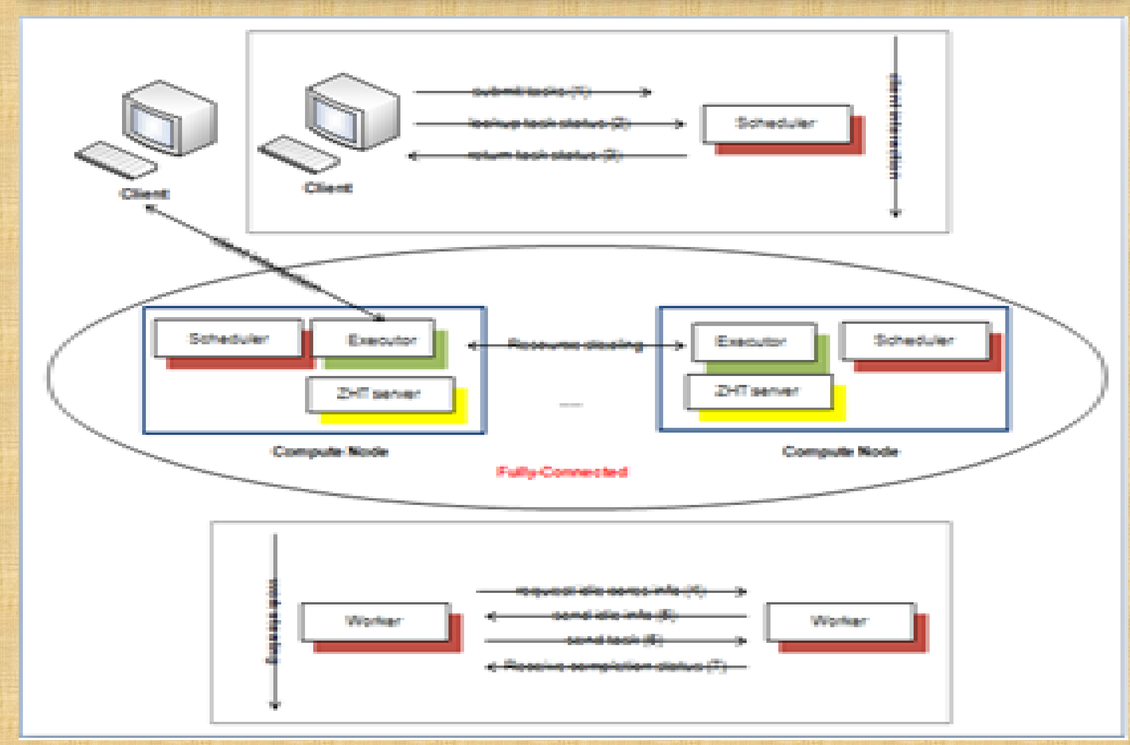
Abstract

- Efficiently scheduling large number of jobs over large scale distributed systems is very critical.
- Today's state-of-the-art job schedulers mostly follow a centralized architecture that is master/slave architecture.
- Aims at providing HPC support on top of MATRIX MTC framework.
- In-corporates resource stealing with work stealing.

Working

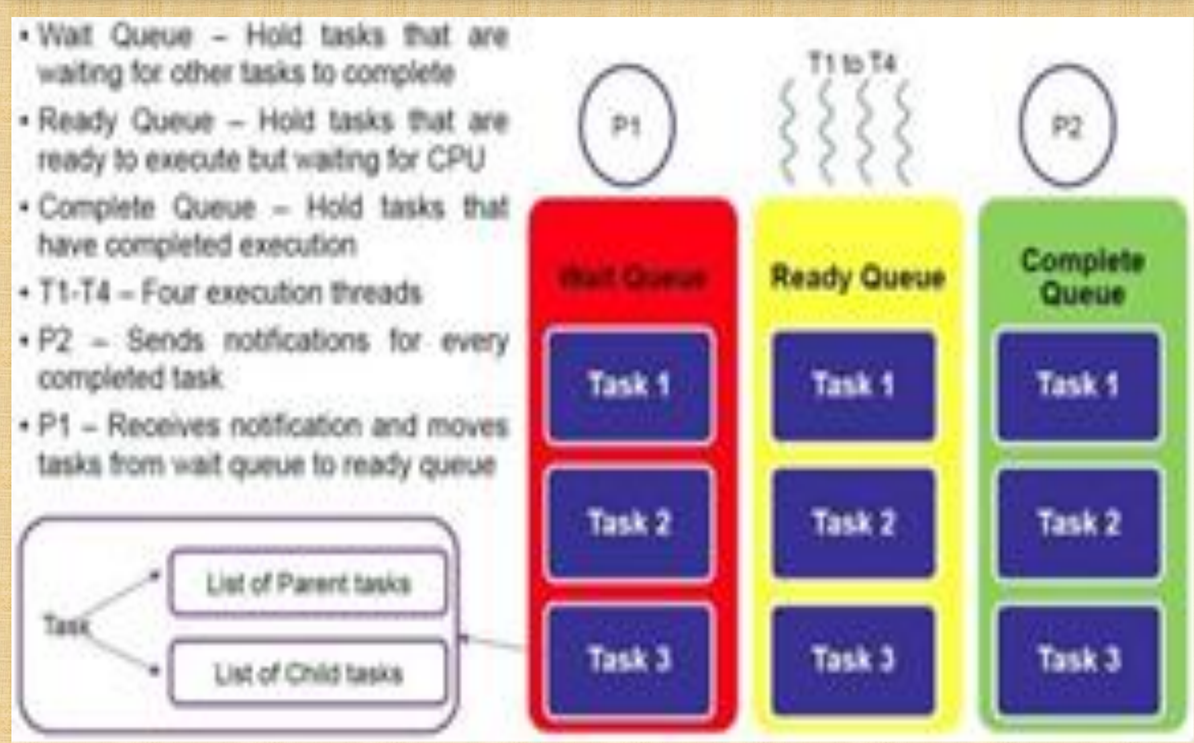
- Scheduler chooses multiple nodes in random.
- Requests for resource information on the nodes.
- Validates if sufficient resources are available to complete the tasks.
- If Yes, Breaks the task into sub-tasks and migrates it to the nodes selected.
- Source receive the results after execution

Architecture

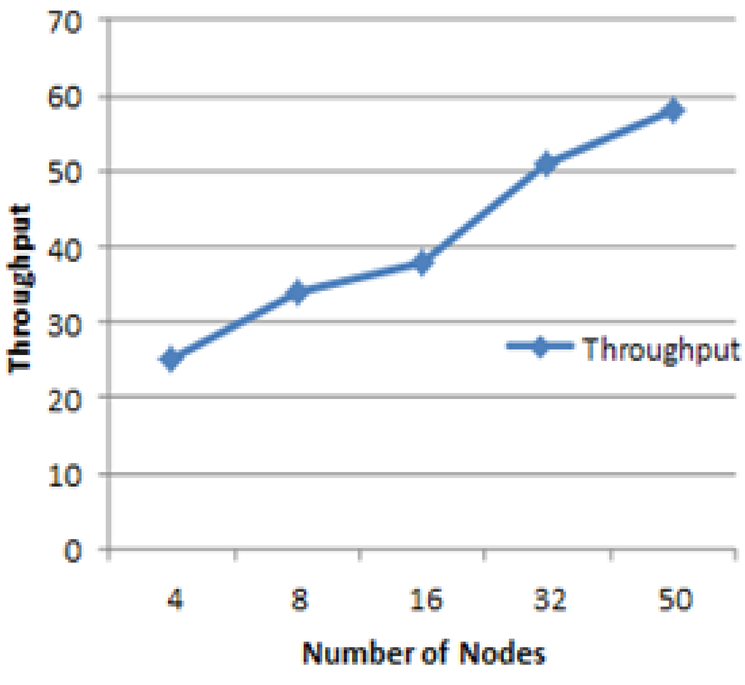


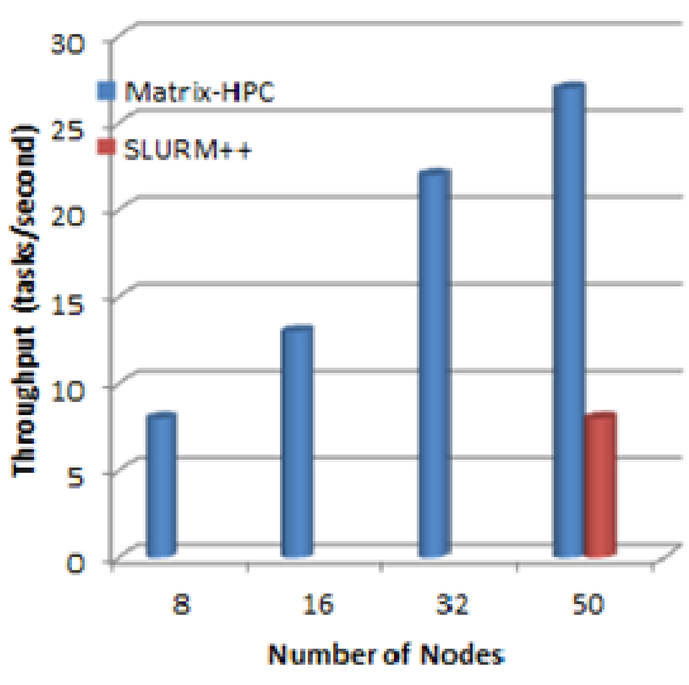
Execution Unit

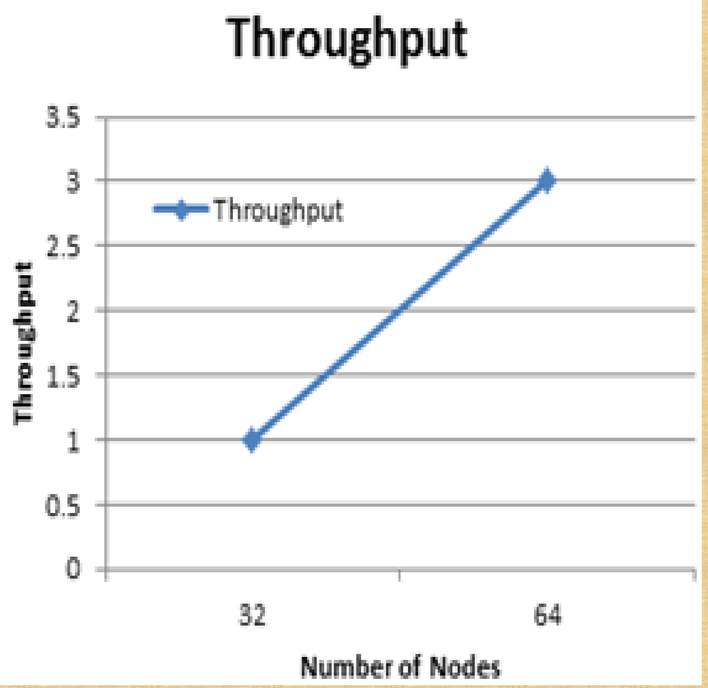
- Wait Queue – Hold tasks that are waiting for other tasks to complete
- Ready Queue – Hold tasks that are ready to execute but waiting for CPU
- Complete Queue – Hold tasks that have completed execution
- T1-T4 – Four execution threads
- P2 – Sends notifications for every completed task
- P1 – Receives notification and moves tasks from wait queue to ready queue



MATRIX-HPC Preliminary Results







Conclusion

- HPC support for MATRIX currently out performs SLURM++ for small task size
- With medium task size, HPC support for MATRIX performs better at 50 nodes.
- With large task size, where each task needs 20 needs HPC support for MATRIX performs better at 50 nodes scale in comparison to SLURM++ at 100 nodes,
- The SLURM++ needs to be run at lesser scales to make a better comparison.

Future Work

- Comparison to be made with SLURM++ at higher scales of 100 nodes and above
- Random neighbor selection will be replaced with history based neighbor selection.
- Resource information can be efficiently maintained by incorporating a key-value pair. This keeps the resource information of each node up to date with less operations.

References

[1] A. Rajendran, I. Raicu. “MATRIX: Many-Task Computing Execution Fabric for Extreme Scales”, Department of Computer Science, Illinois Institute of Technology, MS Thesis, 2013

[2] I. Raicu. "Many-Task Computing: Bridging the Gap between High Throughput Computing and High Performance Computing", Computer Science Department, University of Chicago, Doctorate Dissertation, March 2009