SimMatrix: SIMulator for MAny-Task computing execution fabRIc at eXascales

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

Kevin Brandstatter
Department of Computer Science
Illinois Institute of Technology
kbrandst@hawk.iit.edu

Zhao Zhang
Department of Computer Science
University of Chicago
zhaozhang@uchicago.edu

Ioan Raicu
Department of Computer Science, Illinois Institute of Technology
Mathematics and Computer Science Division, Argonne National Laboratory
iraicu@cs.iit.edu

Abstract

Exascale computers will enable the unraveling of significant scientific mysteries. Predictions are that by 2019, supercomputers will reach exascales with millions of nodes and billions of threads of execution. Many-task computing (MTC) is a new viable distributed paradigm for extreme-scale supercomputing. The MTC paradigm can address four of the five major challenges of exascale computing, namely concurrency, resilience, heterogeneity, and I/O and memory; this work lays the foundations for addressing the first three challenges.

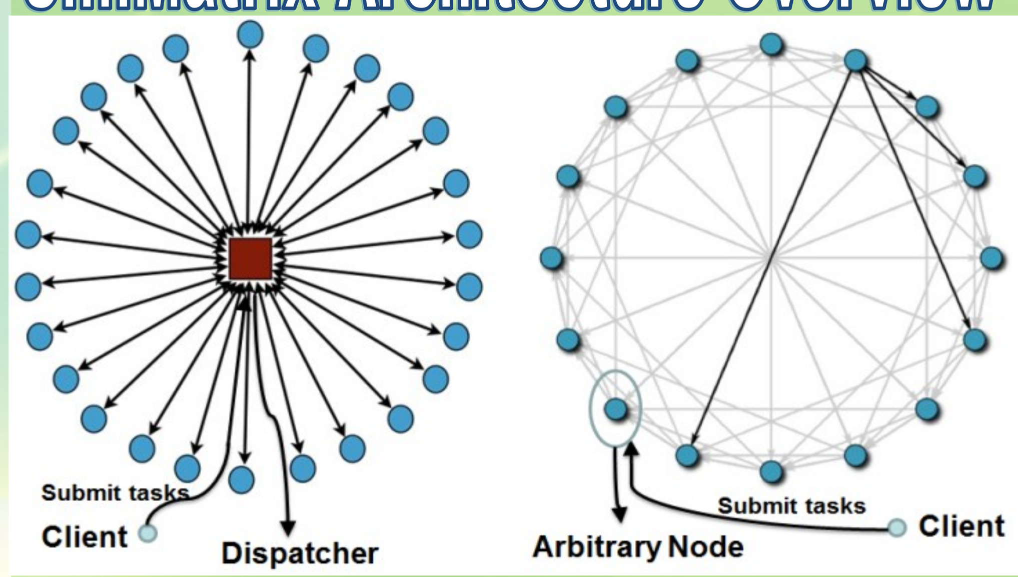
This work presents a new light-weight and scalable discrete event simulator, SimMatrix, which enables the exploration of distributed scheduling for MTC workloads at exascale levels with up to 1 million nodes and 1 billion cores. SimMatrix is validated against a real system, Falkon, with up to 2K-cores, running on an IBM BlueGene/P system. SimMatrix is compared with two other existing simulators, SimGrid and GridSim in terms of scalability and resource (time and memory) consumption. We found that SimMatrix consumes up to 20 bytes less memory per task, and up to 90 us less time per task for distributed scheduling. Due to its excellent scalability, SimMatrix has been able to run at scales up to 1 million nodes, 1 billion cores, and 10 billion tasks with modest resources (e.g. 200GB of memory and 256-core hours).

Work stealing is an efficient distributed load balancing technique whose potential scalability has not been well understood at extreme scales. This work presents an adaptive work stealing algorithm, which is investigated at exascale levels through the SimMatrix simulator. Through SimMatrix, we explore a wide range of parameters important to understand work stealing at up to exascale levels, such as number of tasks to steal, number of neighbors of a node, and static/dynamic neighbors. Experiment results show that adaptive work stealing configured with optimal parameters could scale up to 1 million nodes and 1 billion cores, while achieving 85%+ efficiency running on real MTC workload traces obtained from 17 months from a petascale supercomputer.

Contributions

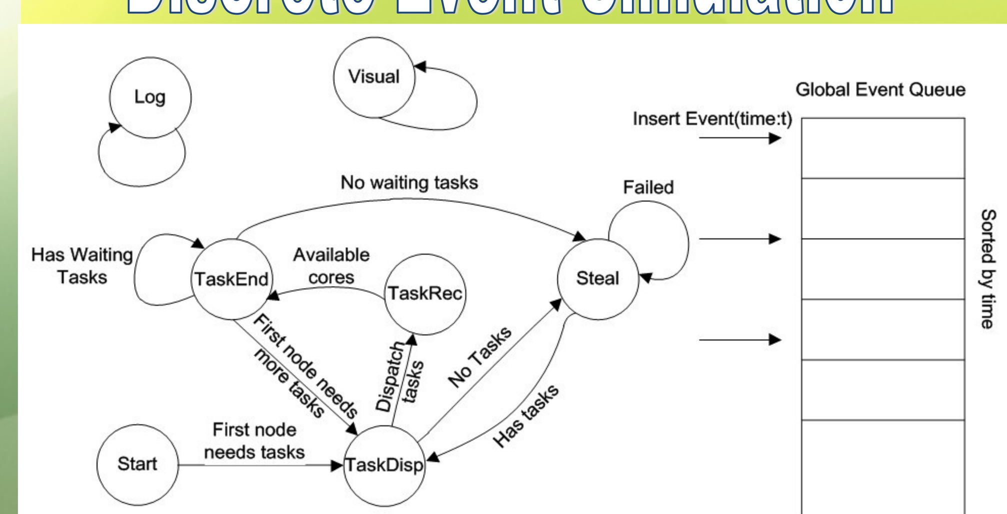
- Develop a new light-weight and scalable discrete event simulator, SimMatrix, which enables distributed scheduling for MTC workloads at exascales. SimMatrix has excellent flexibility and extensibility; it can be used to study both homogenous systems, heterogeneous systems, different programming models (HPC, MTC, or HTC), and different scheduling strategies (centralized, distributed, hierarchical)
- Propose an adaptive work stealing algorithm, which applies dynamic multiple random neighbor selection, and adaptive poll interval techniques.
- Provide evidence that work stealing is a scalable method to achieve distributed load balancing, even at exascales with millions of nodes and billions of cores.
- Identify optimal parameters affecting the performance of work stealing; at the largest scales, in order to achieve the best work stealing performance, we find that the number of tasks to steal is half and there must be a squared root number of dynamic random neighbors (e.g. at 1M nodes, we would need 1K neighbors).

SimMatrix Architecture Overview



The left part is the centralized scheduling with a single dispatcher connecting all nodes; the right part is the homogeneous distributed topology with each node having the same number of cores and neighbors

Discrete Event Simulation



Work Stealing Algorithm

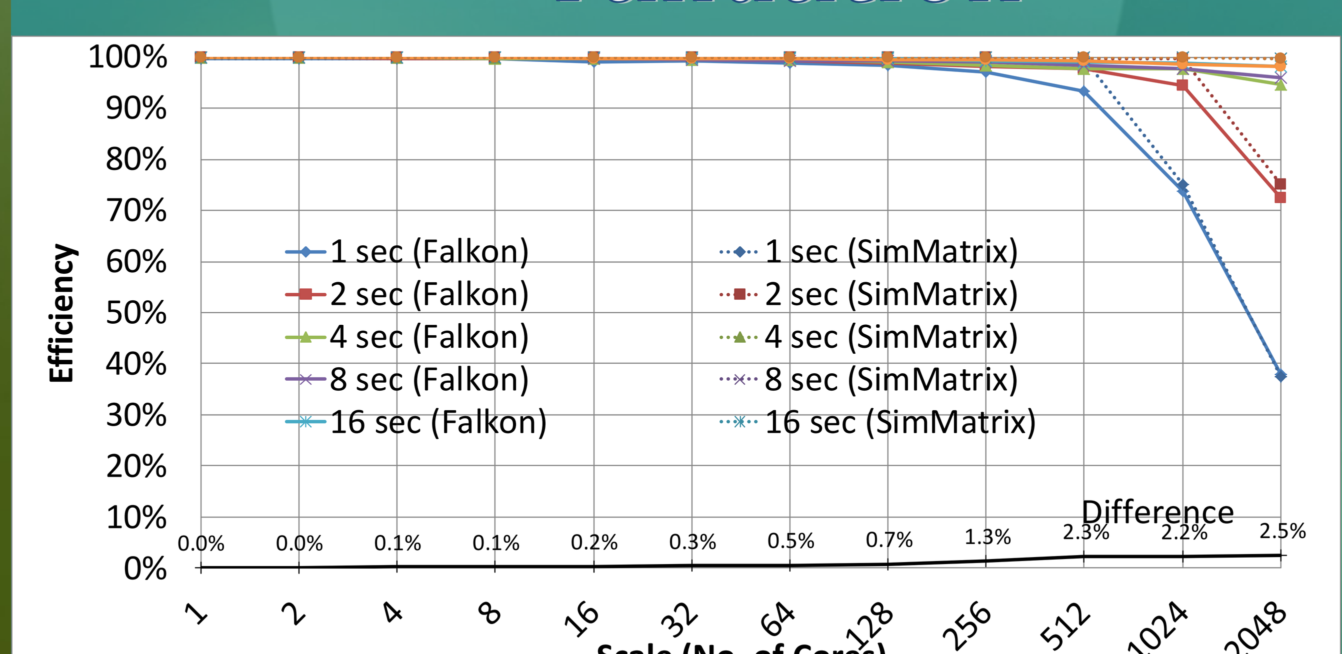
Algorithm 1 Dynamic Multi-Random Neighbor Selection for Work Stealing

DYN-MUL-SEL(*num_neigh, num_nodes*)
- let *selected*[*num_nodes*] be boolean array initialized all false except the node itself
- let *neigh*[*num_neigh*] be array of neighbors
- **for** $i = 1$ **to** *num_neigh*
  - *index* = *random*() % *num_nodes*
  - **while** *selected*[*index*] **do**
    - *index* = *random*() % *num_nodes*
  - **end while**
  - *selected*[*index*] = true
  - *neigh*[*i*] = *node*[*index*]
- **end for**
- **return** *neigh*

Algorithm 2 Adaptive Work Stealing Algorithm

ADA-WORK-STEALING(*num_neigh, num_nodes*)
- *Neigh* = **DYN-MUL-SEL** (*num_neigh, num_nodes*)
- *most_load_node* = *Neigh*[0]
- **for** $i = 1$ **to** *num_neigh*
  - **if** *most_load_node.load* < *Neigh*[*i*].*load* **then**
    - *most_load_node* = *Neigh*[*i*]
  - **end if**
- **end for**
- **if** *most_load_node.load* = 0 **then**
  - *sleep* (*poll_interval*)
  - *poll_interval* = *poll_interval* * 2
  - **ADA-WORK-STEALING**(*num_neigh, num_nodes*)
- **else**
  - steal tasks from *most_load_node*
  - **if** *num_task_steal* = 0 **then**
    - *sleep* (*poll_interval*)
    - *poll_interval* = *poll_interval* * 2
    - **ADA-WORK-STEALING**(*num_neigh, num_nodes*)
  - **else**
    - *poll_interval* = 1
    - **return**
  - **end if**
- **end if**

Validation

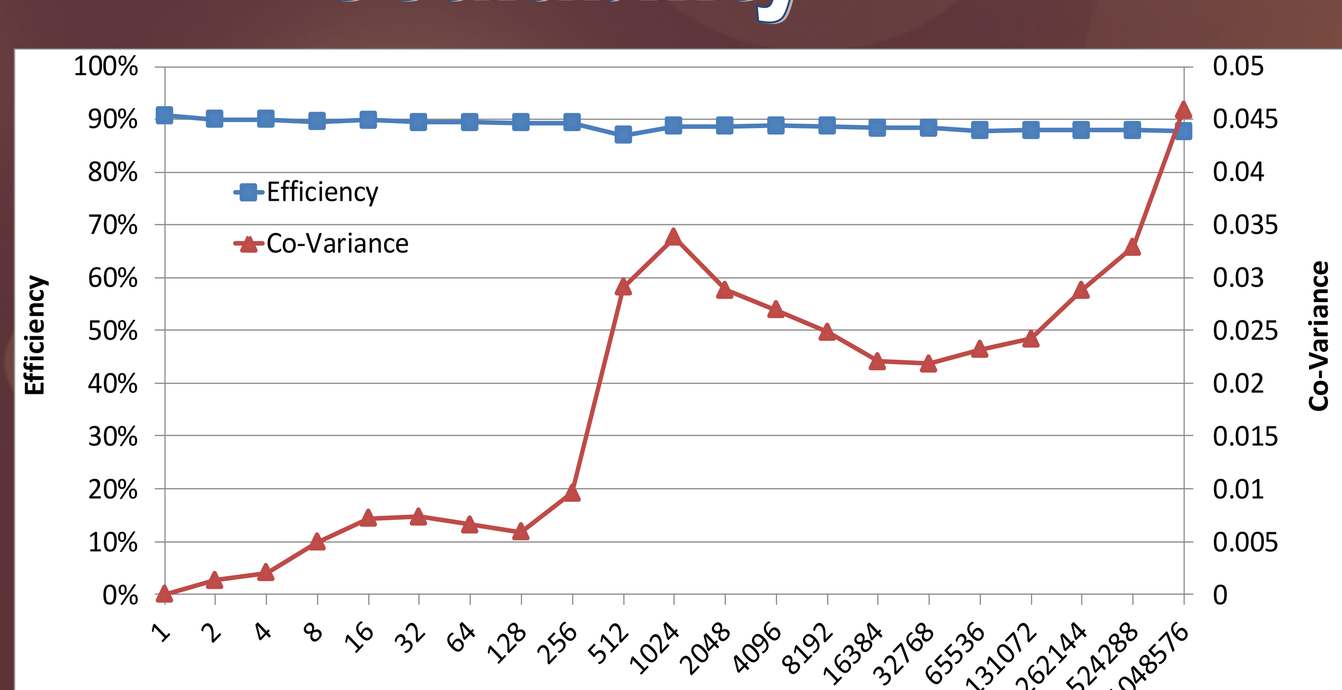


Validate SimMatrix against the state-of-the-art MTC systems (e.g. Falkon), to ensure that the simulator can accurately predict the performance of current petascale systems.

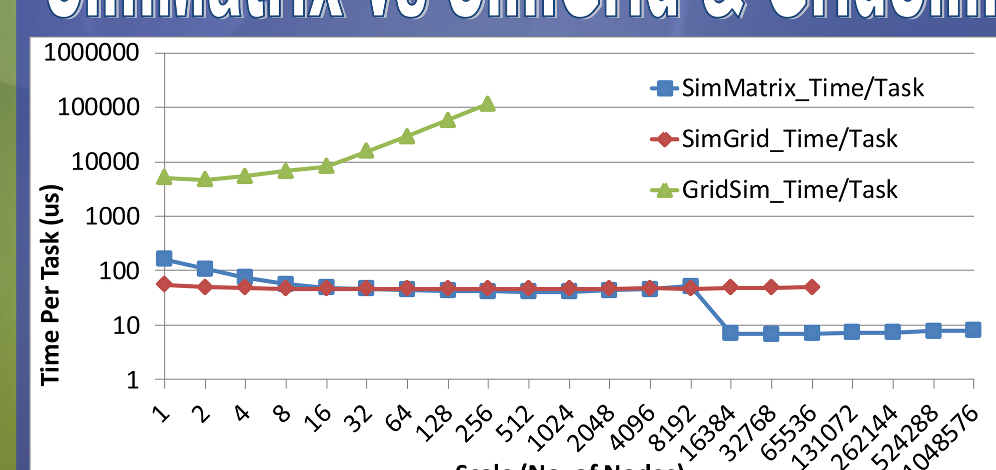
Scalability

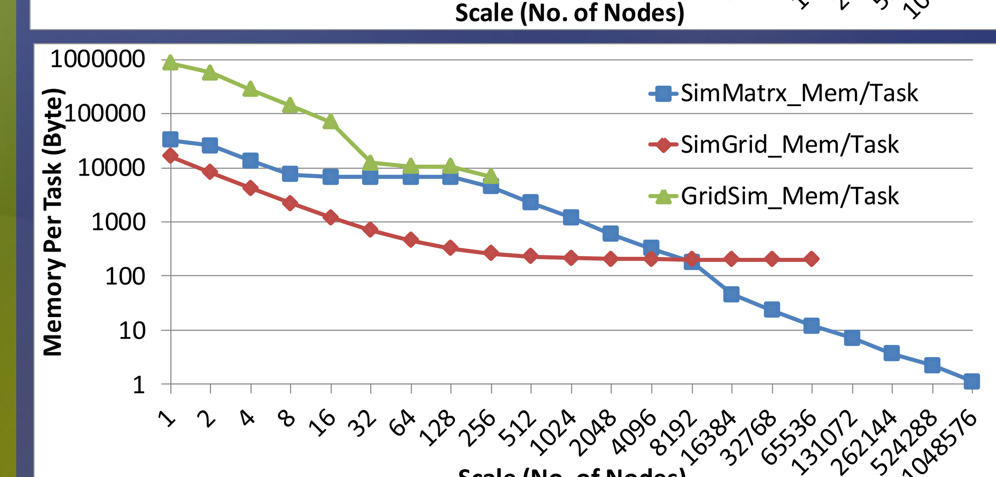
Efficiency 85%+

Co-Variance <0.05 (Load imbalance of <500 tasks from 10K tasks per node)

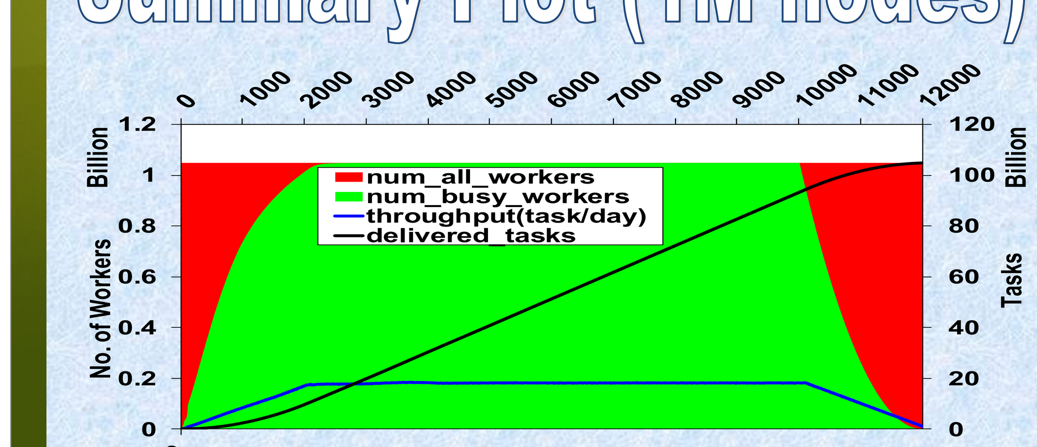


SimMatrix vs SimGrid & GridSim



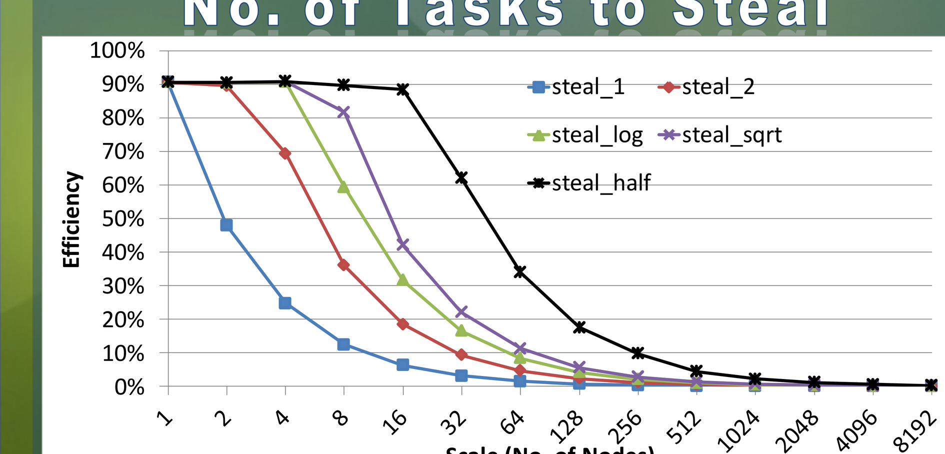


Summary Plot (1M nodes)

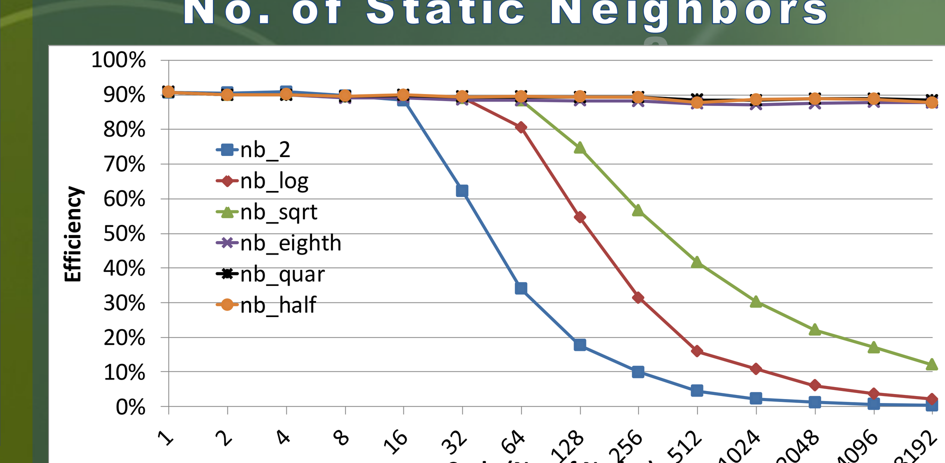


Work Stealing Parameters

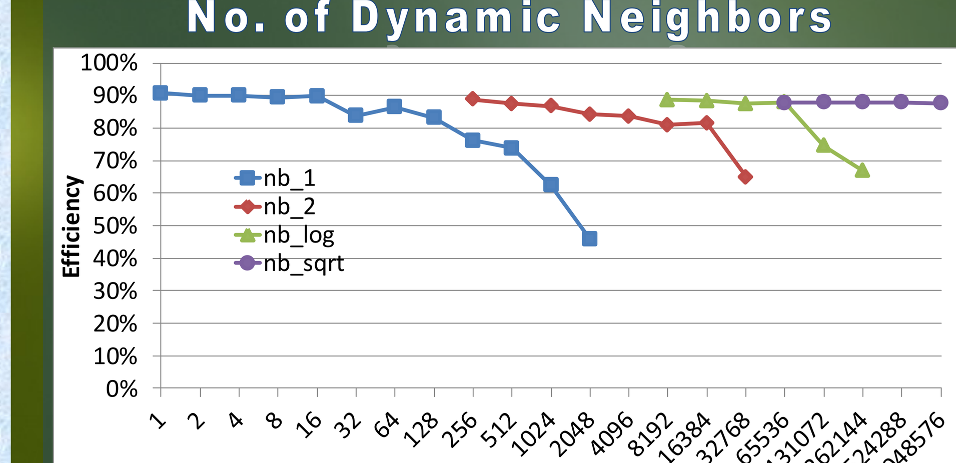
No. of Tasks to Steal



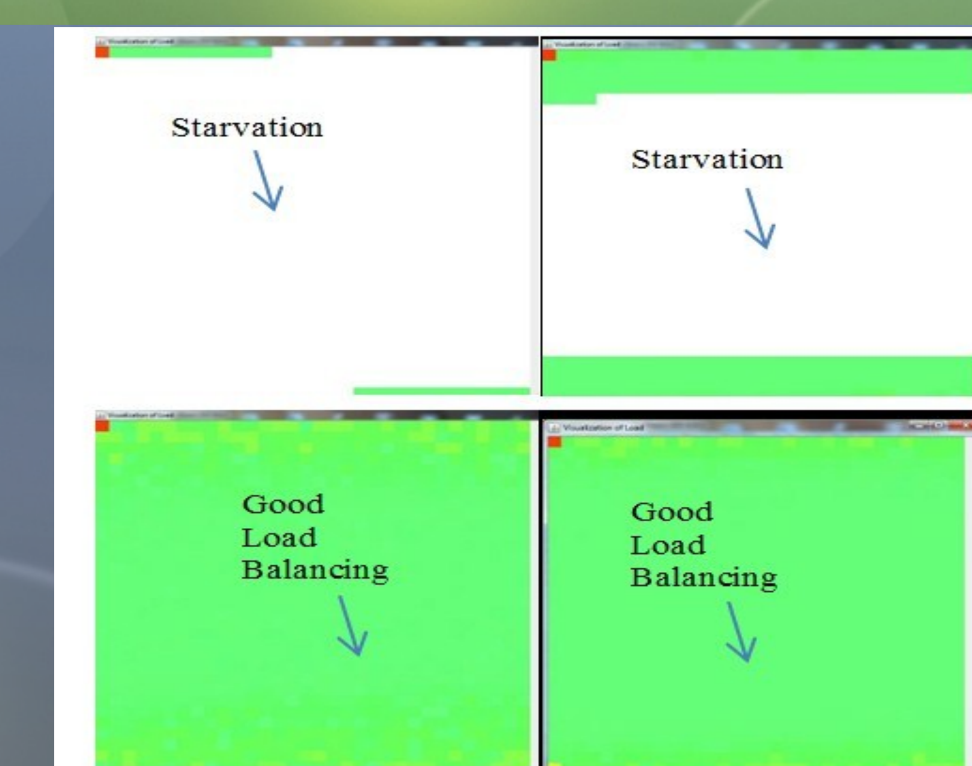
No. of Static Neighbors



No. of Dynamic Neighbors



Visualization



Visualization for 1024 nodes and MTC workload for different number of neighbors; the upper left has 2 static neighbors, the upper right has a squared root static neighbors; the lower left has a quarter static neighbors, the lower right has a squared root dynamic neighbors.

Conclusion & Future Work

Exascale systems bring great opportunities in unraveling of significant scientific mysteries. Also, there are challenges, such as concurrency, resilience, I/O and memory, heterogeneity, and energy. New programming models are needed to solve some of these challenges, and we believe that Many-Task Computing could offer many advantages over High-Performance Computing.

Work stealing is a scalable method to achieve distributed load balance, even at exascales. In order to achieve the best work stealing performance, we find the number of tasks to steal is half and there must be a squared root number of dynamic neighbors (e.g. at 1M nodes, we would need 1K neighbors).

In the future, we will use SimMatrix to explore work stealing for many-core chips with thousands of cores. Another direction for future improvements of SimMatrix is to allow more complex network topologies for an exascale system, such as fat tree, 3D/4D/5D torus networks, daisy chained switches, etc. We will also develop the MATRIX, a MTC task execution fabric at exascales, to implement the proposed adaptive work stealing algorithm. MATRIX will be tested on BlueGene/P/Q systems at full scales, and be integrated with other projects, such as ZHT, FusionFS, Swift, and Charm++.

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

[3] I. Raicu, Y. Zhao, C. Dumitrescu, I. Foster, M. Wilde, "Falkon: A Fast and Light-weight tasK executiON Framework," IEEE/ACM SC 2007.

[4] I. Raicu, I. Foster, et al, "Middleware Support for Many-Task Computing," Cluster Computing, The Journal of Networks, Software Tools and Applications, 2010.