Paving the Road to Exascales with Many-Task Computing

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

Anupam Rajendran
Department of Computer Science
Illinois Institute of Technology
arajend5@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 (with millions of nodes and billions of cores) will enable the unraveling of significant scientific mysteries around the year 2019. Many-task computing (MTC) is a new viable distributed paradigm for extreme-scale supercomputing. The MTC paradigm can address three of the four major challenges of exascale computing, namely Concurrency and Locality; Resiliency; Memory and Storage;

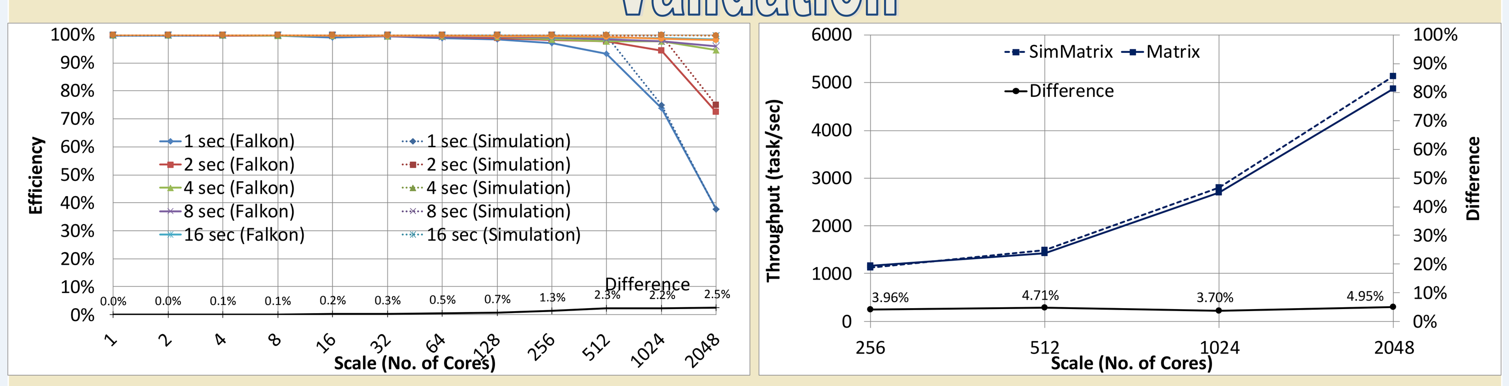
SimMatrix is a new light-weight and scalable discrete event simulator, which serves as the simulator for MTC execution fabric at exascales. It supports both the centralized and distributed scheduling. Work stealing is an efficient distributed load balancing technique. Through SimMatrix, we explore a 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. SimMatrix is validated against Falkon for FIFO centralized scheduling, and against MATRIX for work stealing based distributed scheduling, using MTC workloads up to 2K-cores on a BlueGene/P supercomputer. Simulation results demonstrate that work stealing configured with optimal parameters has the potential to scale to exascales, while achieving 85%+ efficiency under real MTC workload traces obtained from a 17-month period on a petascale supercomputer. In addition, 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 less than 1 bytes, 10 us per task for centralized scheduling, and 20 bytes, 90 us per task for distributed scheduling at scales up to 1 million nodes, 1 billion cores, and 10 billion tasks. 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).

MATRIX is a distributed many-Task computing execution framework, which utilizes the adaptive work stealing algorithm to achieve distributed load balancing. MATRIX uses ZHT (a distributed zero hop key-value store) for task metadata management, to submit tasks and monitor the task execution progress. We have a functional prototype implemented in C, and have scaled this prototype on a BG/P supercomputer up to 512-nodes (2K-cores) with good results.

Contributions

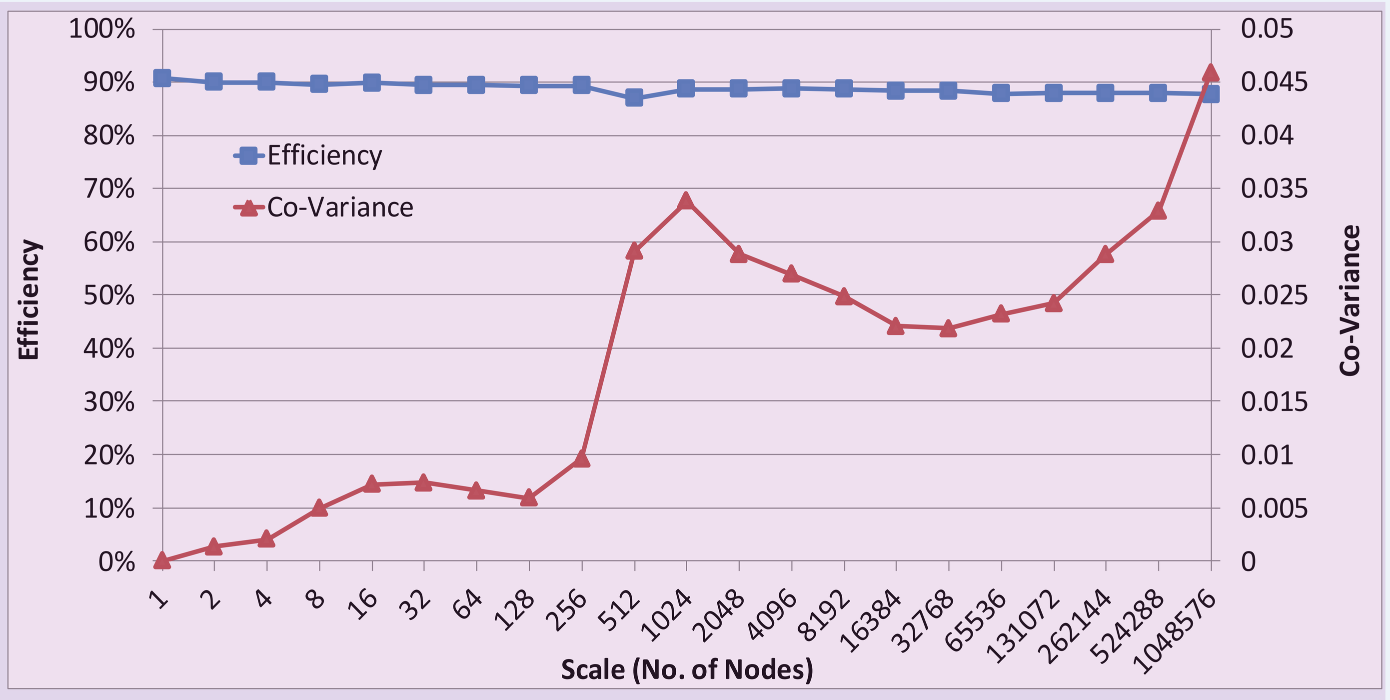
- Develop a new light-weight and scalable discrete event simulator, SimMatrix, which enables distributed scheduling for MTC workloads at exascales. SimMatrix has excellent flexibility and extensibility; it can be used to study both homogenous systems, heterogeneous systems, different programming models (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).

Validation



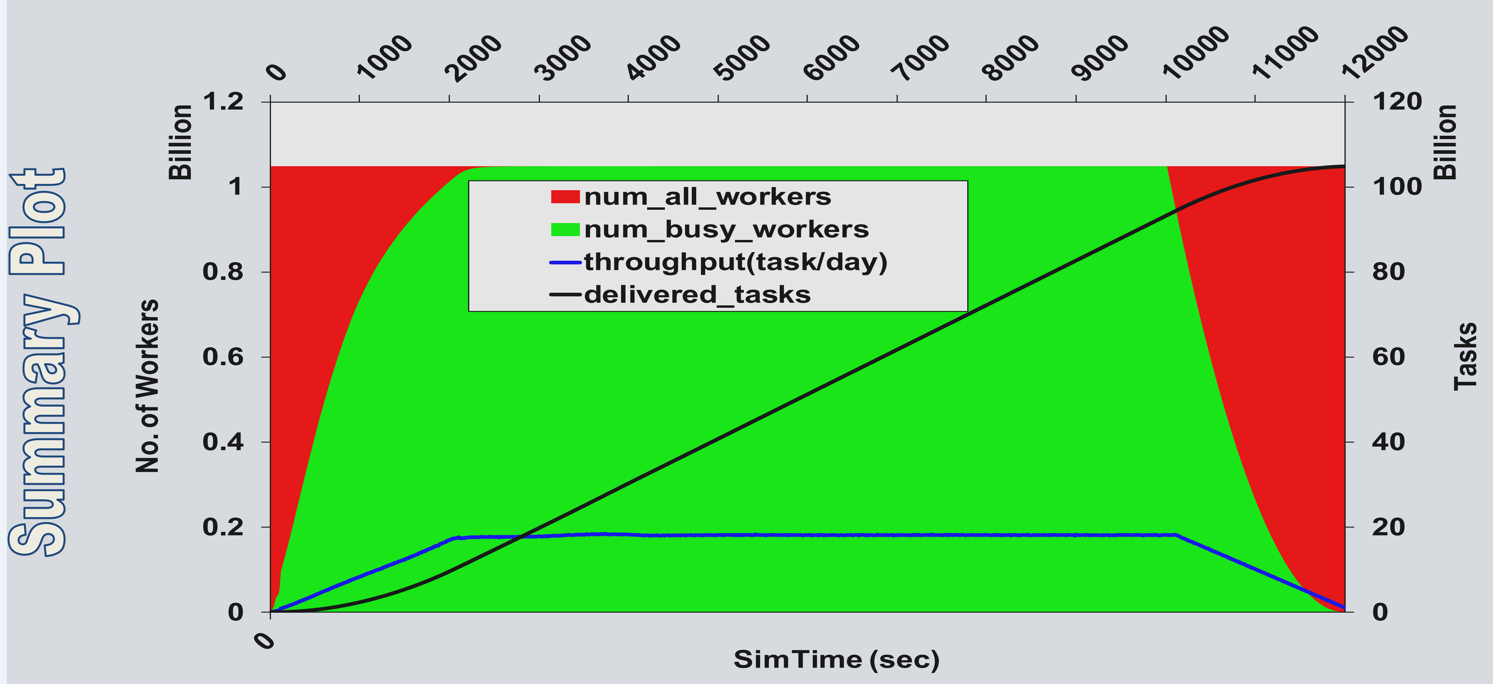
Left is the validation of SimMatrix against Falkon with up to 2K cores for different sleep workloads; right is the validation of SimMatrix against MATRIX with up to 2K cores for sleep 0 tasks. Falkon and MATRIX were run on BG/P

Scalability

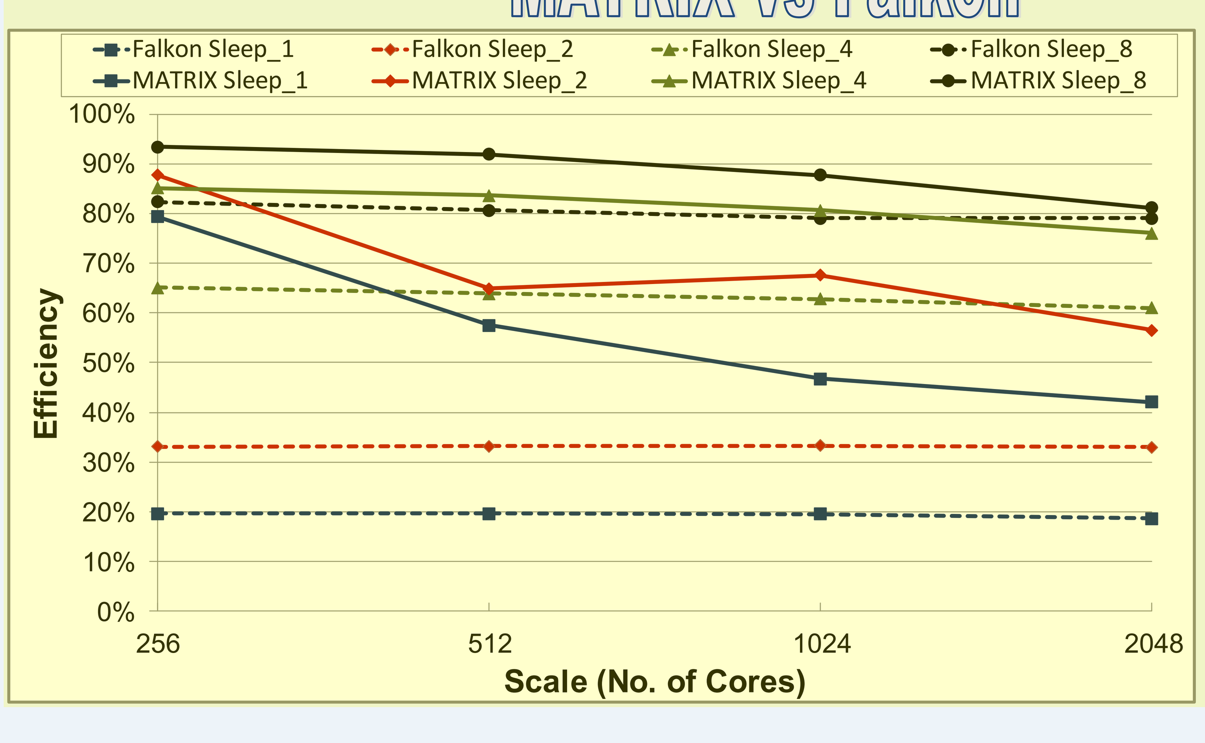


Scalability of distributed scheduling with work stealing; at extreme scale, work stealing achieves 87%+ efficiency

Summary Plot

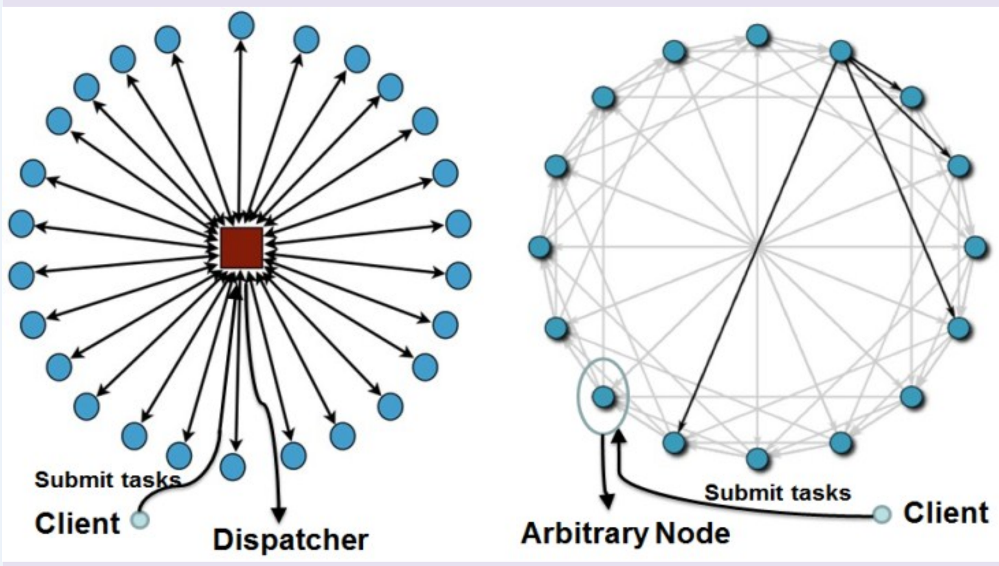


MATRIX vs Falkon



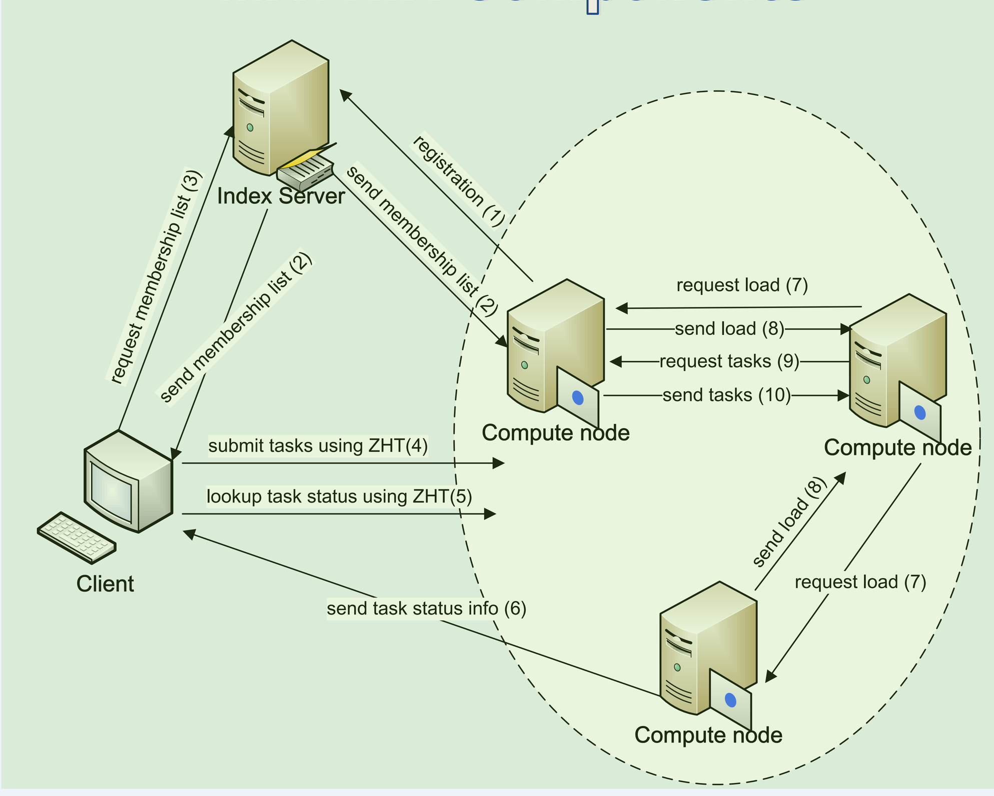
Comparison of MATRIX with Falkon with different sleep workloads at the scales up to 2K cores. MATRIX implemented work stealing as the distributed scheduling strategy; while Falkon implemented a hierarchical scheduling strategy with different levels of dispatchers. All experiments were run on BG/P machine

SimMatrix Architecture

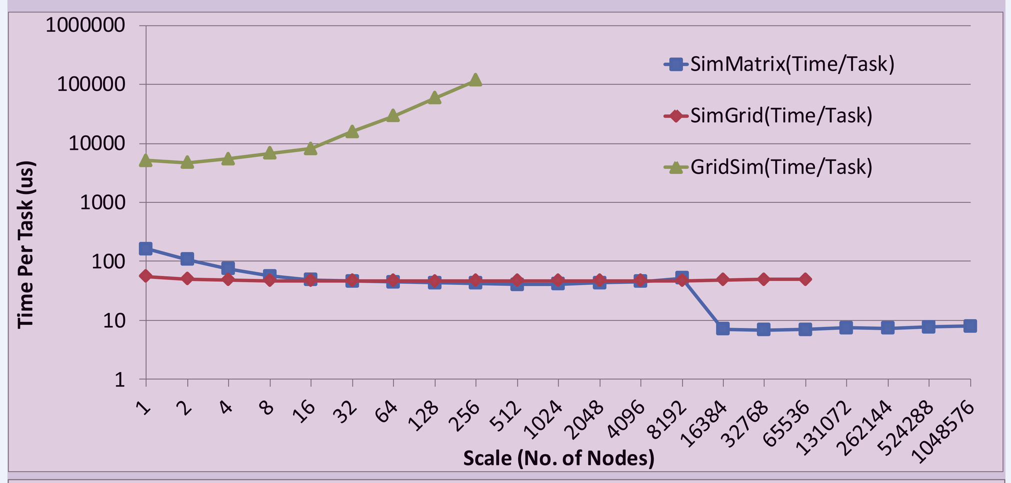


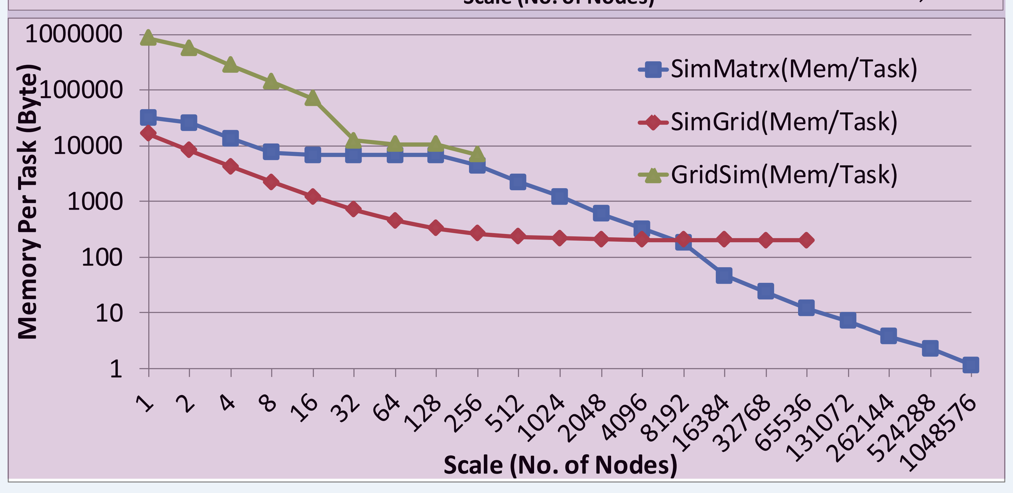
Left is the centralized scheduling with a single dispatcher connecting all nodes; right is the homogeneous distributed topology with each node having the same number of cores and neighbors

MATRIX Components



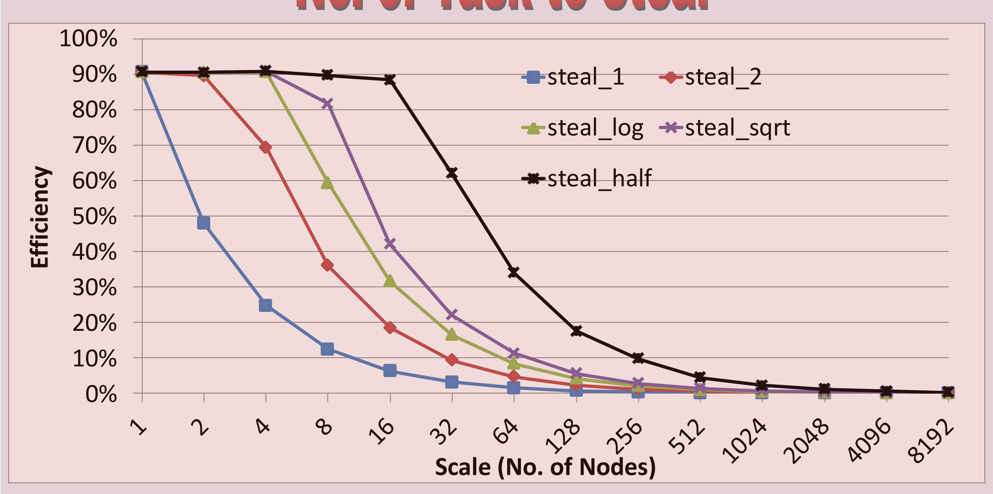
SimMatrix vs SimGrid & GridSim



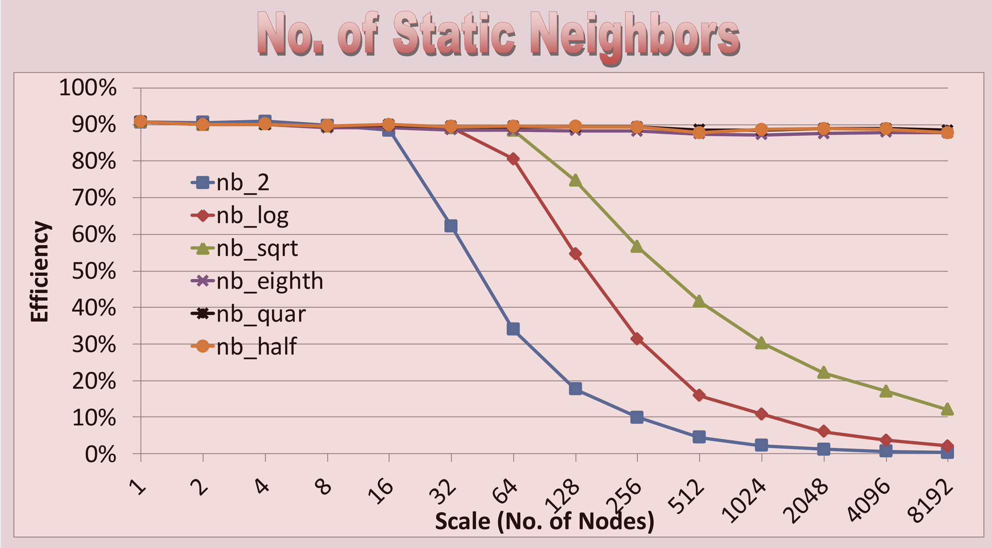


Work Stealing Parameters

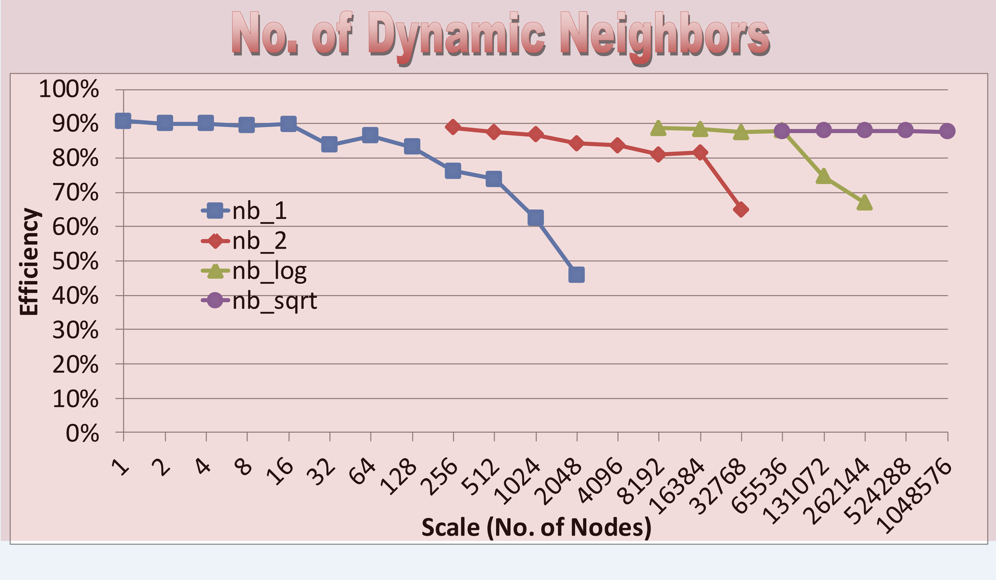
No. of Task to Steal



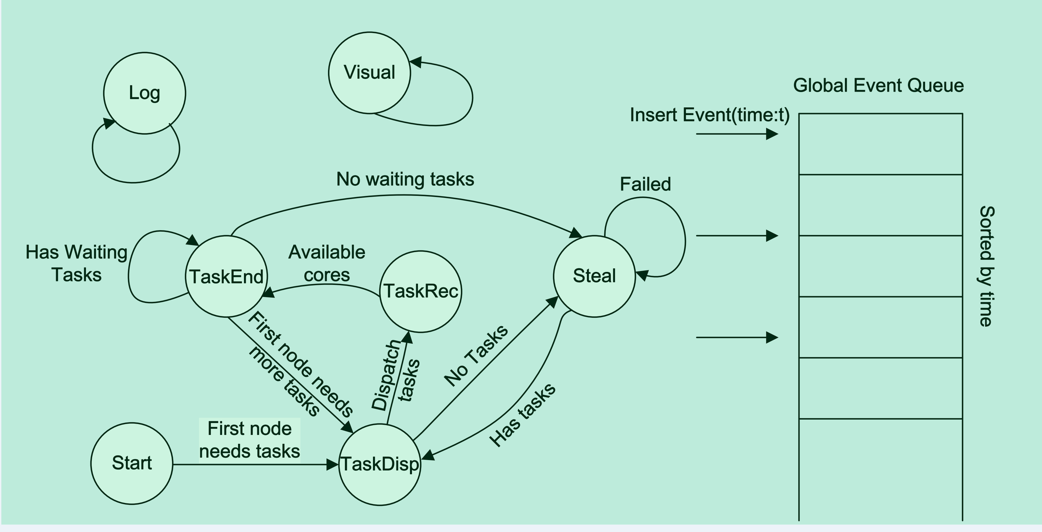
No. of Static Neighbors



No. of Dynamic Neighbors



Discrete Event Simulation



Work Stealing

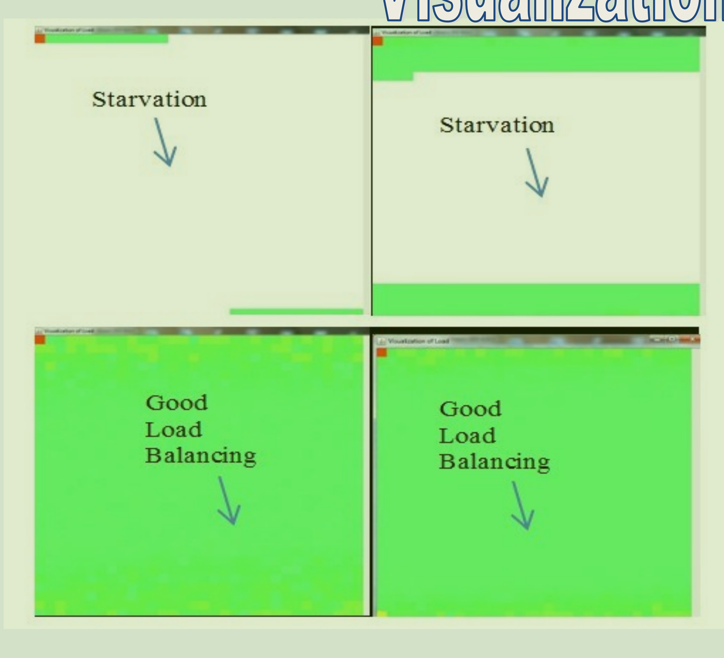
Algorithm 1 Dynamic Multi-Random Neighbor Selection for Work Stealing

```
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
```

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 will bring great opportunities in unraveling of significant scientific mysteries. Also, there are challenges, such as Energy and Power; Memory and Storage; Concurrency and Locality; Resiliency. Any one of these challenges, if left unaddressed, could halt progress towards exascale computing. New programming models are needed to solve some of these challenges, and we believe that Many-Task Computing (MTC) could offer many advantages over High-Performance Computing (HPC).

Work stealing is a scalable technology to achieve distributed load balancing, even at the extreme scale with millions of nodes and billions of cores. 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. Also, we will implement task dependency and workflow simulation in SimMatrix. 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 continue to develop the MATRIX, which will be tested on BG/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.