Centralized and Distributed Job Scheduling System Simulation at Exascale

Speakers: Ke Wang

Advisor: Ioan Raicu

Coworker: Juan Carlos Hernández Munuera

09/07/2011

Outline

- Introduction
- Simulated Architecture
- Technical Consideration
- Centralized Simulator
- Distributed Simulator
- Conclusion and Future Work

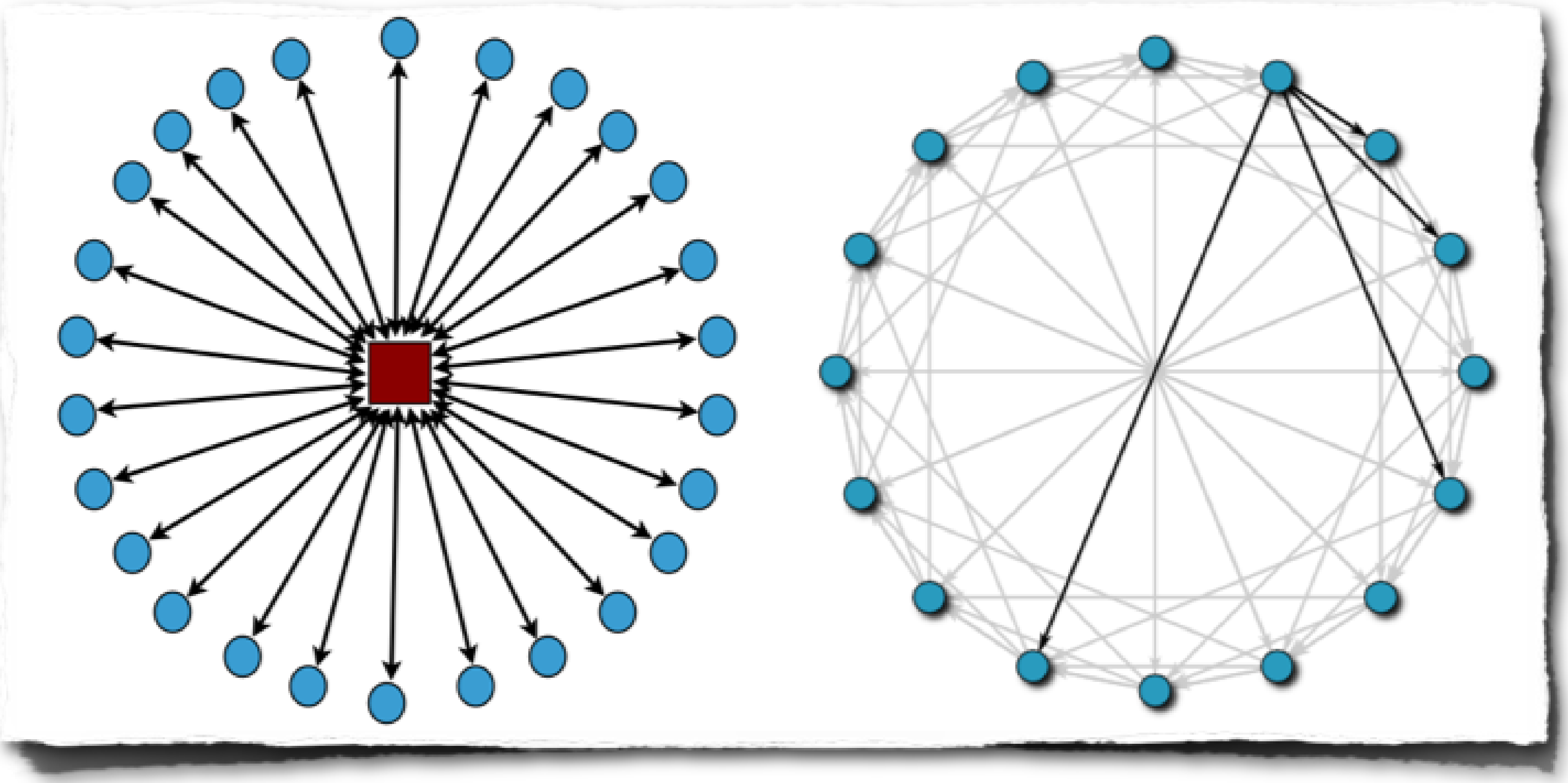
Introduction

- MTC
- Job Scheduling Systems
- State-of-art Job Scheduling Systems and Simulators(Centralized/Small scale)
- Exascale

Our Work

- Study scalability and feasibility of JOB SCHEDULING at EXASCALES
- Simulation
  - memory and processing limitations
  - realistic representation of real systems
- Explore central and decentralized systems
- Carry experiments to draw useful conclusions

Simulated Architecture



Technical Consideration

- Hardware, simulating demands great resources
  - Fusion, 48 cores, 64 GB memory
  - Thread limitation! 1:1 mapping discarded
- Software
  - Simulation model, discrete or continuous-events
  - Existing simulation environments
    - GridSim, SimJava, JiST

JiST

- Java in Simulation Time
- Incredibly light for a simulation environment
  - naive ring of million nodes just 1.3 GB
- Easy discrete-event abstraction
- Centralized simulator developed
- Work discontinued
  - Own semantics, debugging
  - Undetermined execution order of events at the same time
  - Weird errors, JiST not support anymore

Centralized Simulator

- Components
- Client
- Server
- Nodes
- Event Queue
- Load information Hash Map
- How the simulator works?

Centralized Simulator Implementation

- 1. Global Variables

Variables	Description
int numNode	Number of nodes the simulator would have
double linkSpeed	The link speed of the network
double procTimePerJob	Time the server takes to determine which node to dispatch for one job
double networkLatency	Network latency for every communicate message
int numCoresPerNode	Number of cores each node has
double jobSize	The size of each job
int lowThreshold	The threshold to which point the client submits more jobs to the server
long totalNumJobs	Number of jobs the client need to do
double logTimeInterval	The time interval to write log

Centralized Simulator Implementation

- 2. Job Waiting Queue in the Centralized Server

- Data Structures and the time efficiency

Data Structures	**Removing from the head**	**Adding from the rear**
Vector	$\Theta(n)$	$\Theta(1)$
ArrayList	$\Theta(n)$	$\Theta(1)$
LinkedList	**$\Theta(1)$**	**$\Theta(1)$**

Centralized Simulator Implementation

- 3. Event Queue
- (1) Stores events that will happen in future
- (2) Each event has an attribute of occurrence time
- (3) The first event in the event queue is the one that has smallest occurrence time

Centralized Simulator Implementation

- 3. Event Queue
- Event type and description

Event Type	Description
JobEnd	A job is finished by a node. Has other fields: ‘jobKey’, ‘nodeKey and ‘timeStamps’
Submission	Client submits some number of jobs to the centralized server
Log	Write a record to the log file at the simulation time

Centralized Simulator Implementation

- **3. Event Queue**
  - Update frequently. All operations need to maintain the event queue
  - Heapsort: takes Θ(lgn) time for removing and inserting and Θ(1) time for getting the first element
  - In java, TreeSet is a set whose elements are ordered using their natural ordering, or by a comparator provided at set creation time. Implemented based on Red-Black tree, guaranteeing Θ(lgn) time for removing and inserting and Θ(1) time for getting the first.

Centralized Simulator Implementation

- 4. Load Information

  (1) load = number of busy cores, the range of load is [0, numCoresPerNode]

  (2) Using Hash Map to store load information

  <Key, Value>, Key = load, Value is a hashset containing the node ids which have the load.

  Update the hashmap takes $\Theta(1)$ time

Centralized Simulator Implementation

- 5. Logs and Plot Generation
- Two logs
- task_execute_log: records information such as the ‘submission time’, ‘wait time’, ‘executing time’ for every job. Has switch to turn it on/off.
- summary_log: contains information such as ‘number of all cores’, ‘number of executing cores’, ‘waiting queue length’, ‘through put’. Implement it with an event instead of separate thread.
- Six ways to write to a log: ‘FileOutputStream’, ‘BufferedOutputStream’, ‘PrintStream’, ‘FileWriter’, ‘BufferedWriter’, ‘PrintWriter’. ‘BufferedWriter’ is the fastest one.
- Use ploticus to generate plots

Results and Discussions

- 1. Values of global variables for experiments

Variables	Values
linkSpeed	1000000000 bytes/Sec
procTimePerJob	1 millisecond
networkLatency	100 microseconds
numCoresPerNode	1000
jobSize	1000 bytes
lowThreshold	2000

Results and Discussions

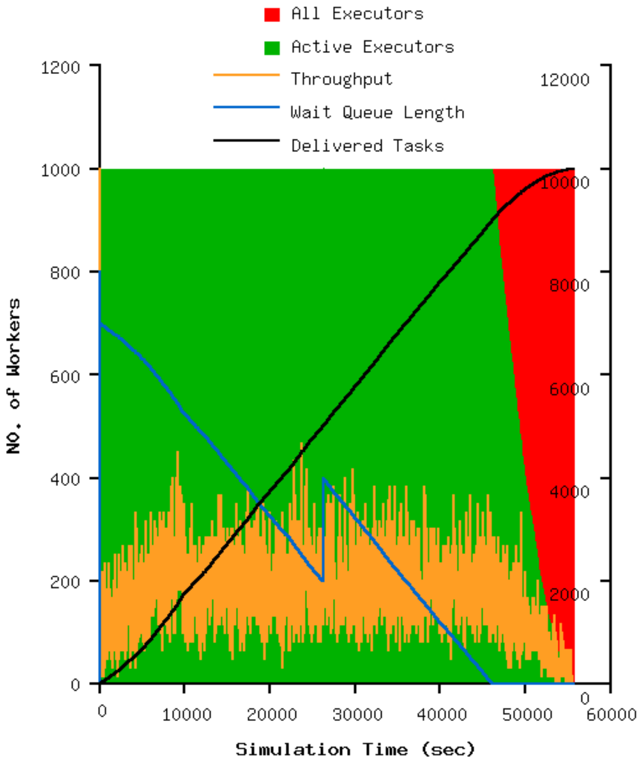
- 2. Correction Validation
- communication overhead is 0, procTimePerJob = 0, networkLatency = 0, jobSize = 0. totalNumJobs is 10 times of the total number of cores. Two groups of experiments.
- (1) all the jobs have the same length, 1000 seconds. Simulation time is: 1000 * 10 = 10000
- (2) the average length of all jobs is 500 seconds. Simulation time is around 500 * 10 = 5000
- These two results are exactly what we expect.

Performance Results

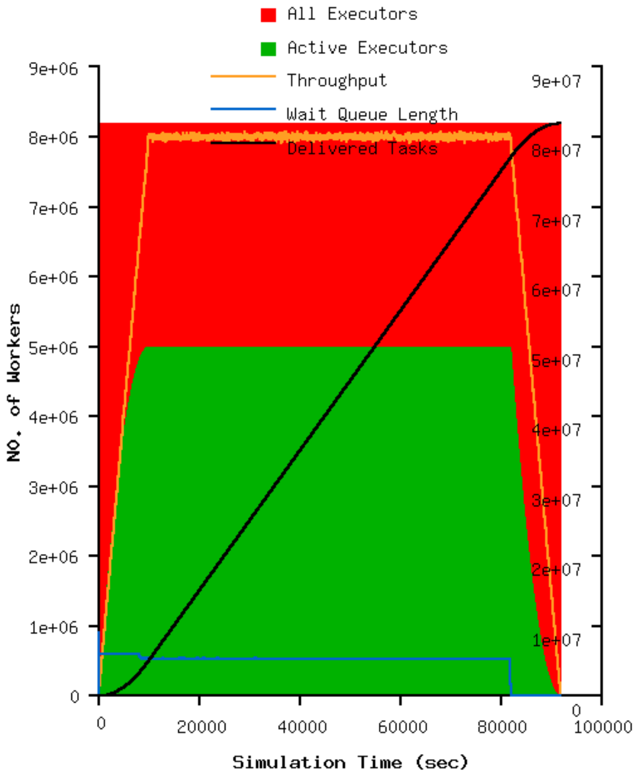
	Average Job Length: 5000 seconds		Average Job Length: 500000 seconds	
No. of Nodes	Simulation Time(s)	Real Time(s)	Simulation Time(s)	Real Time(s)
1	55711.5063	1.302	5626603.597	1.386
2	56623.62122	1.56	5645321.744	1.807
4	56447.07415	2.154	5636848.561	2.314
8	56569.38075	3.171	5673423.825	3.615
16	56682.29121	4.929	5661830.737	5.553
32	56686.82232	8.367	5659072.997	9.169
64	56724.64275	16.001	5668874.294	16.533
128	56673.38761	31.021	5667498.57	30.855
256	56788.76278	57.157	5664111.183	55.868
512	56928.37883	115.36	5664150.113	110.426
1024	57196.01807	237.705	5665329.831	223.32
2048	57773.9418	500.294	5667987.182	470.484
4096	59156.70364	1463.532	5668656.772	1344.491
8192	91915.50152	398.35	5670328.713	3334.99
16384	173831.1205	857.332	5674380.508	6818.076
32768	337677.7676	1882.023	5682968.991	14089.804
65536	665353.5454	3721.908	5699856.595	33116.881
131072	1320718.623	8175.676	5735198.21	111091.141
262144	2631434.502	16148.014	/	/
524288	5252877.111	29795.722	/	/
1048576	10495753.63	67251.93	/	/

Plots

Performance of 1 node, average job length is 5000 seconds, multiply the throughput by 10000

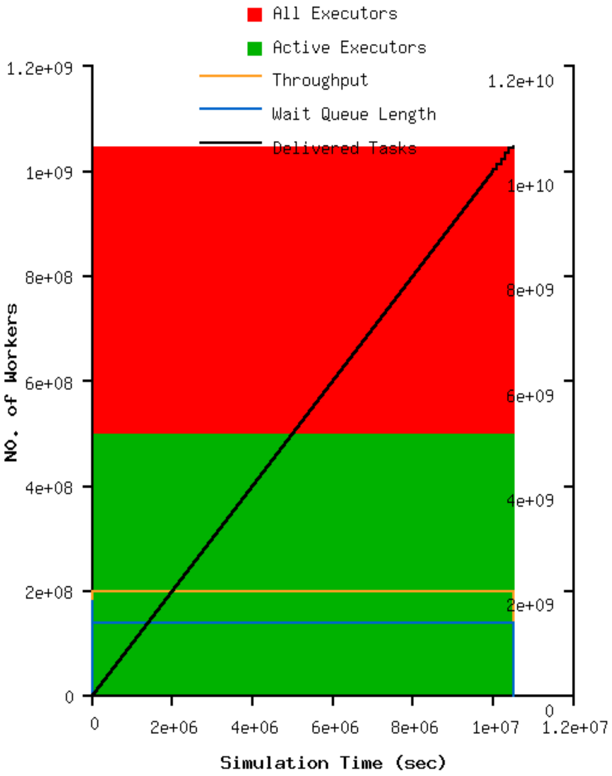


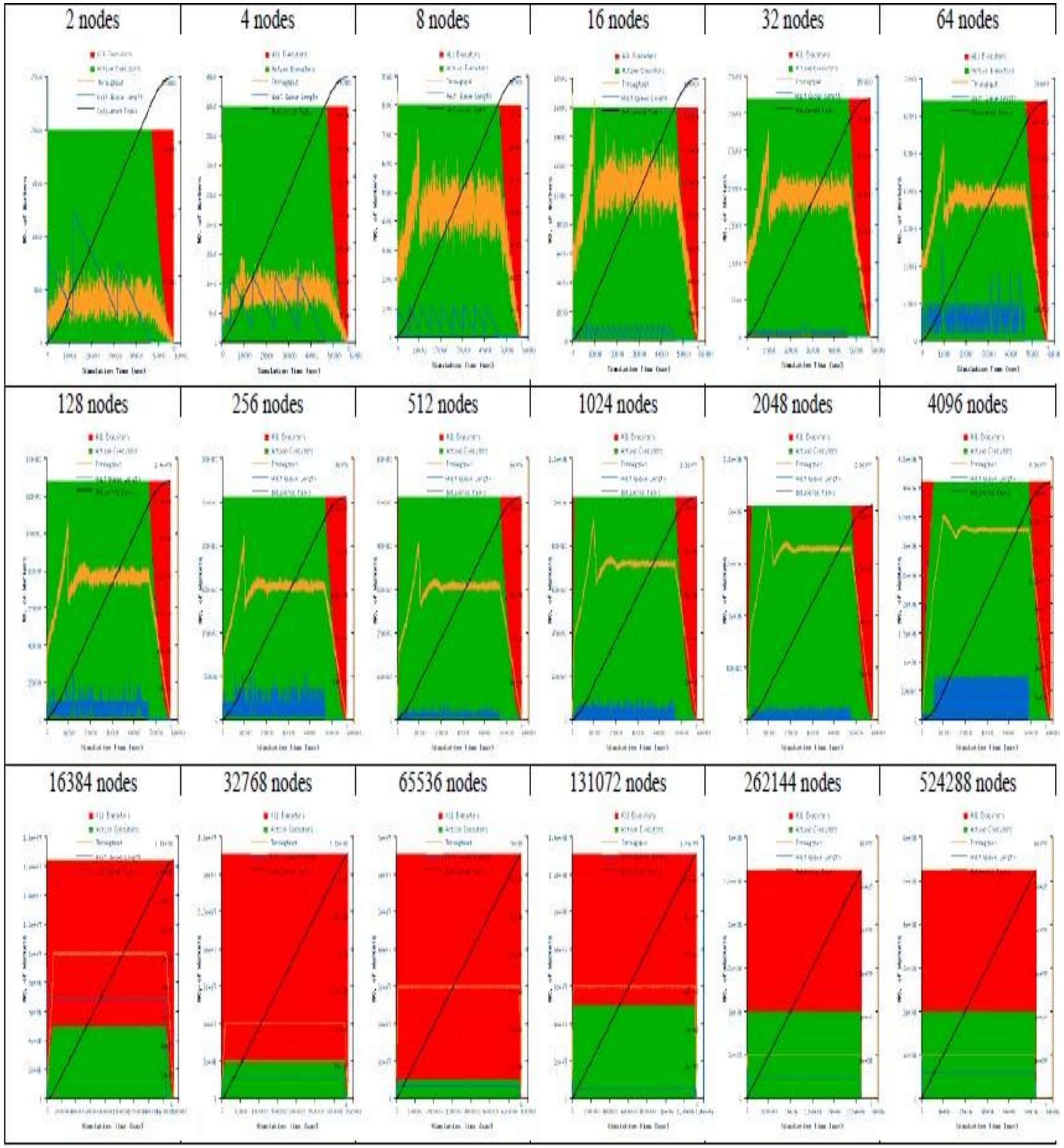
performance of 8192 node, average job length is 5000 seconds, multiply the throughput by 80000, the wait queue length by 1000



Plots

performance of 1048576 node, average job length is 5000 seconds, multiply the number of executed cores by 100, the throughput by 2000000 and the wait queue length by 200000





Distributed Simulator

- Improve the throughput and reliability
- Load balancing is trivial for centralized simulator
- Implement work stealing to achieve load balancing

Work Stealing

- An efficient method to achieve load balancing
- Processes have load imbalance at first. Many benchmarks to generate load imbalance, such as BPC(Bouncing Producer-Consumer), UTS(Unbalanced Tree Search)
- The idle processes poll the busy ones to get work to do.
- Thief: The process that initiates the steal
- Victim: the process that is targeted by the steal

Work Stealing

- Parameters affecting the performance of work stealing
- Can a node steal jobs from all others or just some neighbors?
- How to define neighbors?
- How to select which neighbor to steal jobs
- How many jobs to steal from a selected node?

Changes from the Centralized Simulator

- (1) Remove the centralized server and enhance the functionality of a node.
- (2) A node has a few number of neighbors from which it could steal or dispatch jobs. consider just homogeneous network, that is the distances between a node and its neighbors are the same.
- (3) Keep the global event queue except more events
- (4) Handle jobs straightforwardly, no job entity.
- (5) Client just submits to the first node.
- (6) load = jobListSize – numIdleCores
- (7) Do visualization for the load for every node
- (8) Termination condition: all jobs submitted by client are finished

Distributed Simulator Implementation

1. Global Variables for work stealing and visualization

Variables	Descriptions
int numberNeighbors	How many number of nodes each node has
long currentEventKey	Event key for an event
int numNeigToAsk	How many number of neighbors to ask during one attempted stealing
double noJobsToStealLongInterv	The poll interval of a node to ask jobs when it is idle
double noJobsToStealShortInterv	The time interval of a node to ask another part of its neighbors when it is idle
int numStealWork	How many jobs to steal from a neighbor
double visualizationInterval	The time interval to do visualization

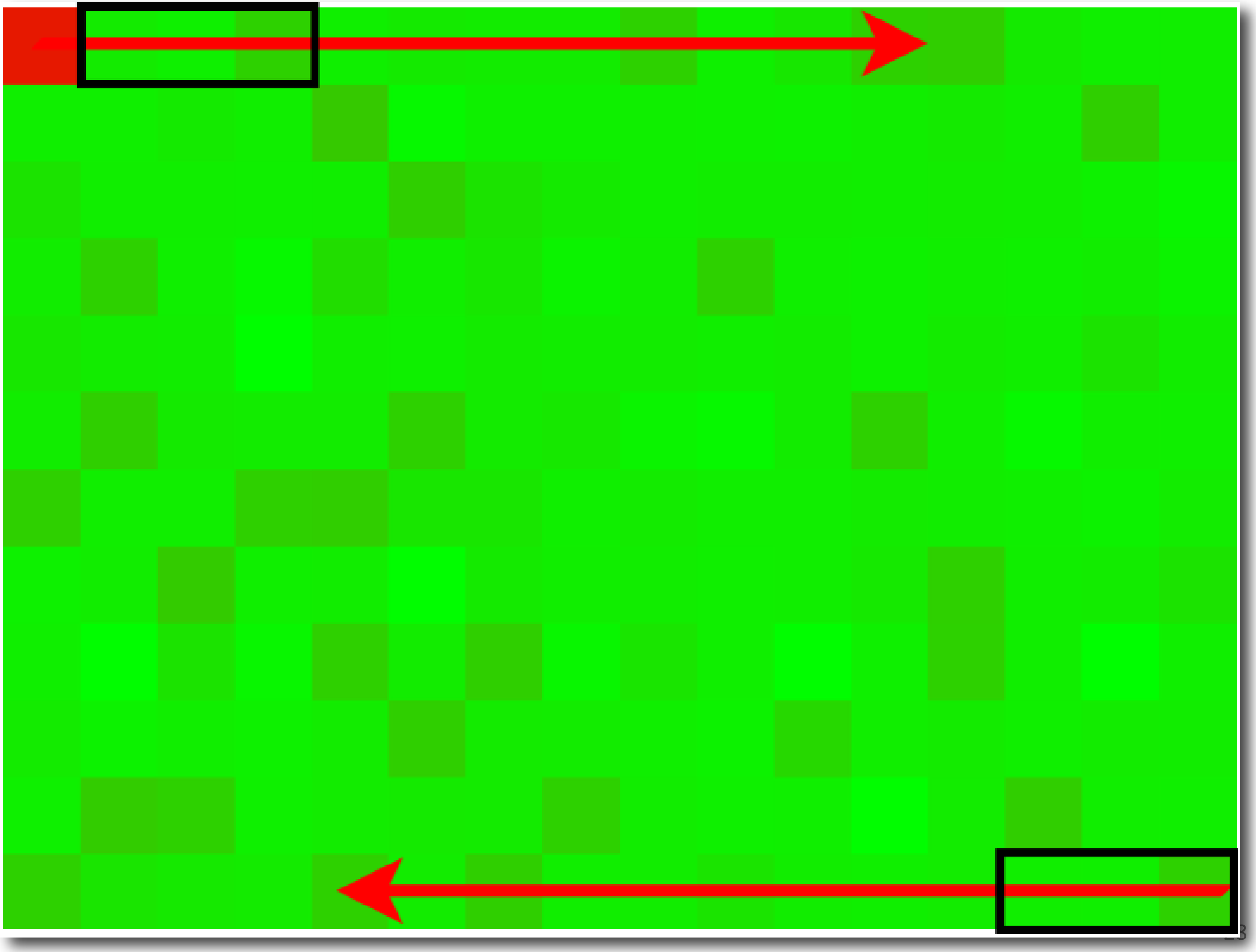
2. Global Event Queue

- Each event now has a global id number
- Types of descriptions of events

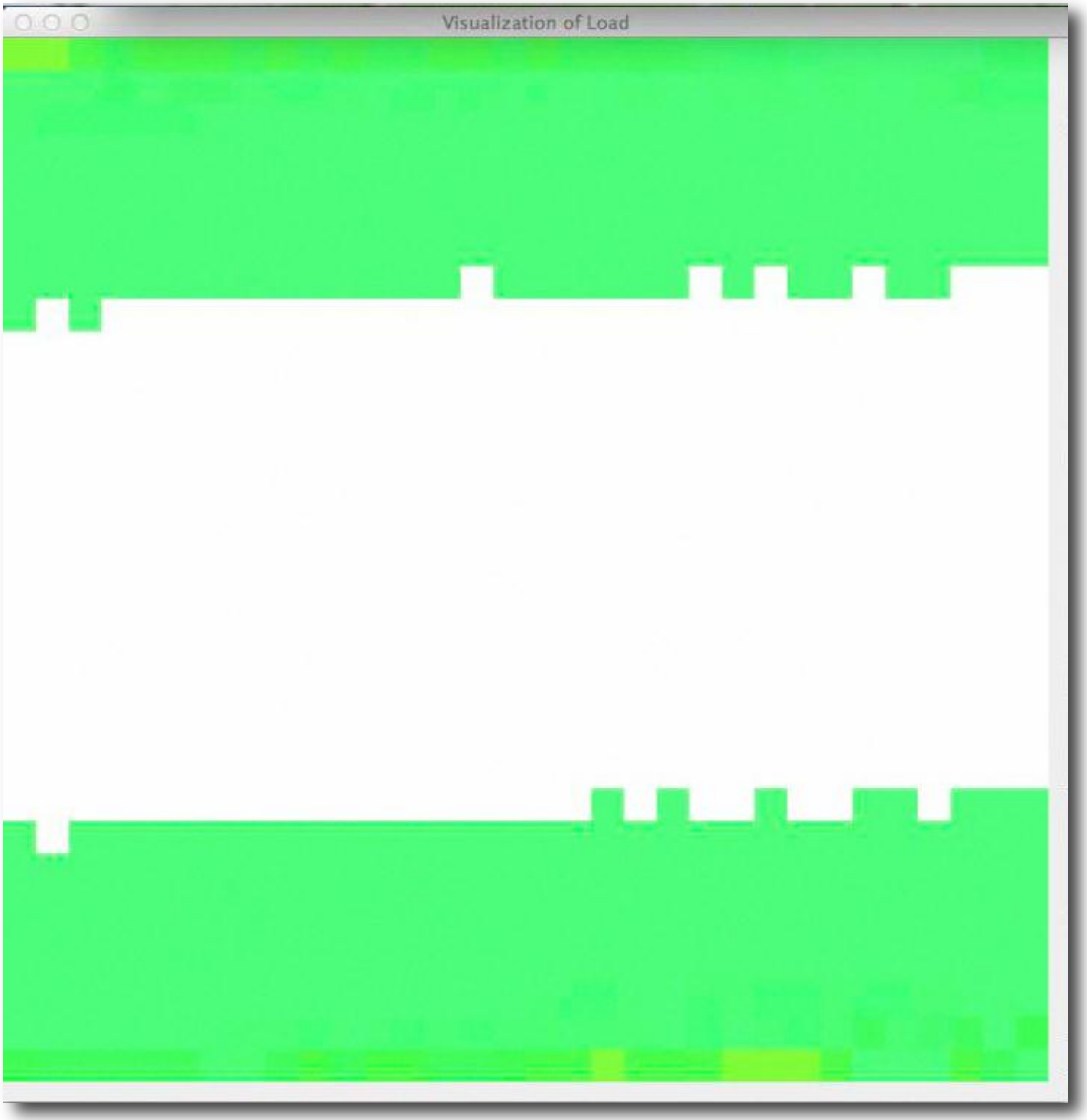
Event Type	Event Description
JobEnd	A job is finished, a cores is free. Start to execute another job or steal jobs
Steal	Ask jobs from its neighbors. Ask load, choose heaviest and inserts 'JobReception', or wait for some time to ask again.
JobDispatch	A node dispatches jobs to a neighbor. Has jobs, inserts 'JobReception' from the neighbor, or ask the neighbor do steal again
JobReception	First node receive jobs from client, or a node receive jobs form its neighbor
Log	The same as that of centralized simulator, add coefficient variance
Visualization	Visualize the load information of all node

3. Visualization

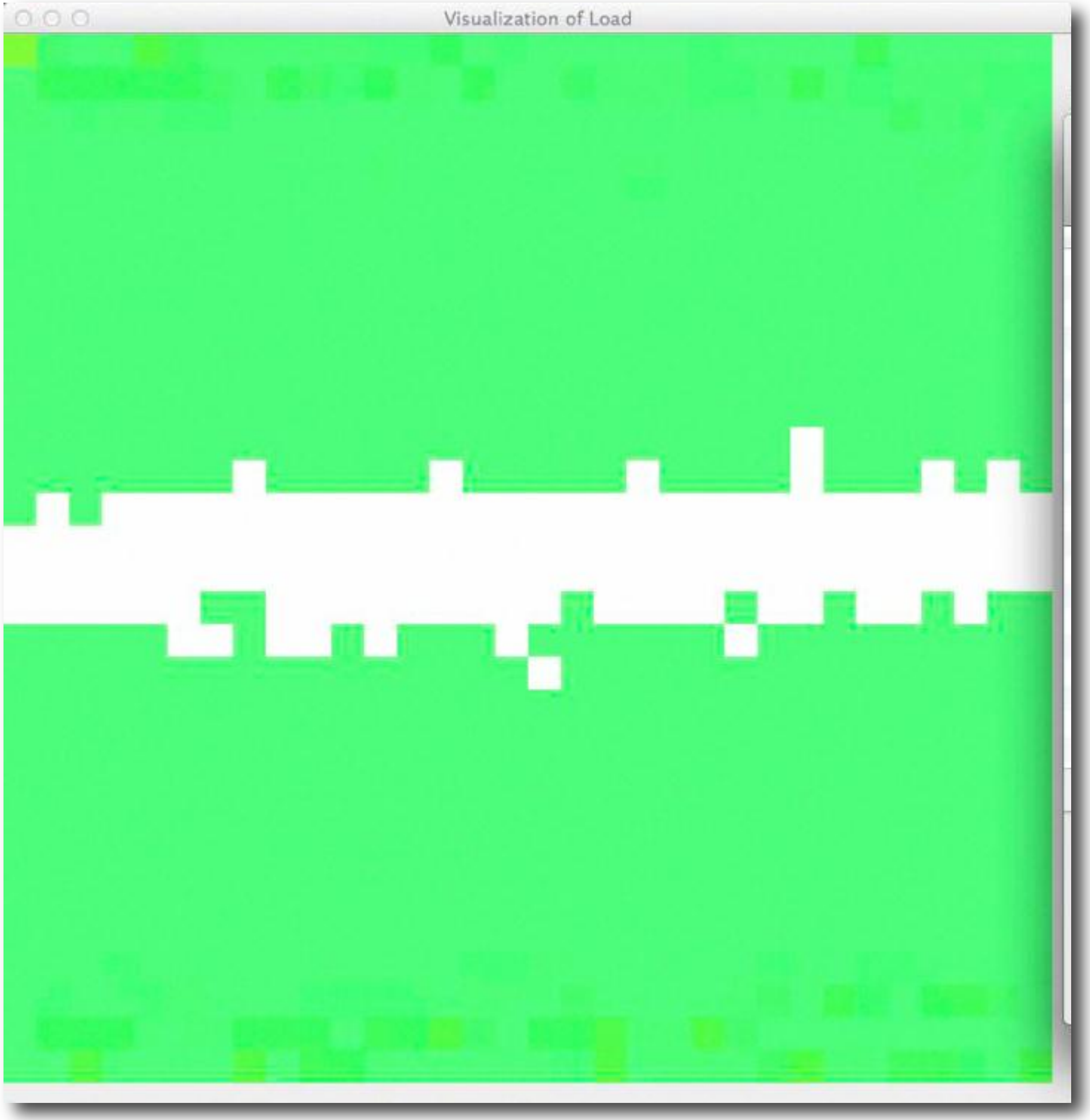
- Efficiently represent load flow in the system
- Simple canvas, each node mapped to a tile
  - color represent load
  - Best results in HSB color space
    - Hue = (1-rate)*0.36
    - Brightness = 1.0
    - Saturation = 1.0-(0.4*(1-rate))



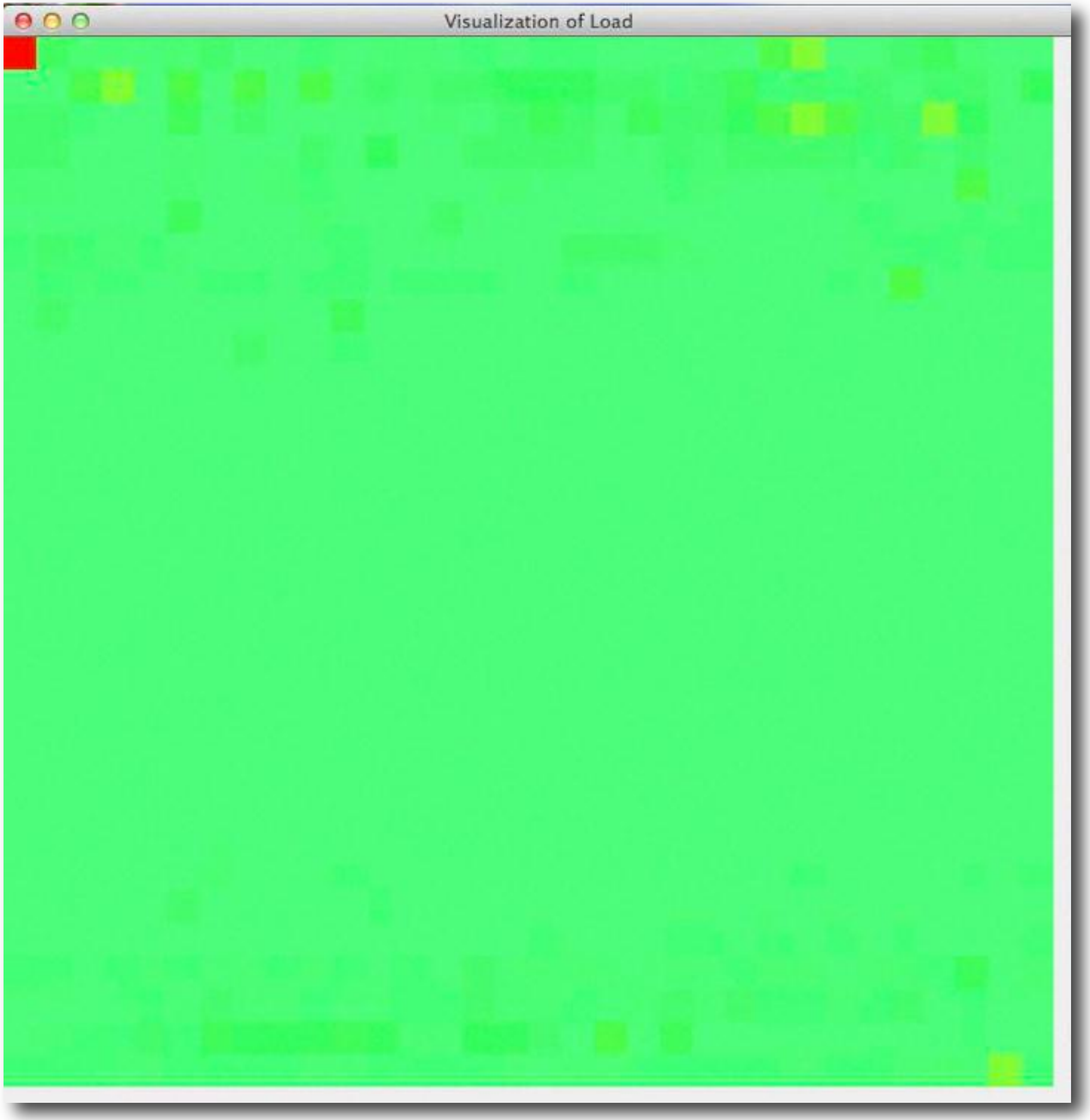
1024 nodes, 8 cores, 64 neighbors, 100000000 jobs



1024 nodes, 8 cores, 128 neighbors, 100000000 jobs



1024 nodes, 8 cores, 256 neighbors, 100000000 jobs

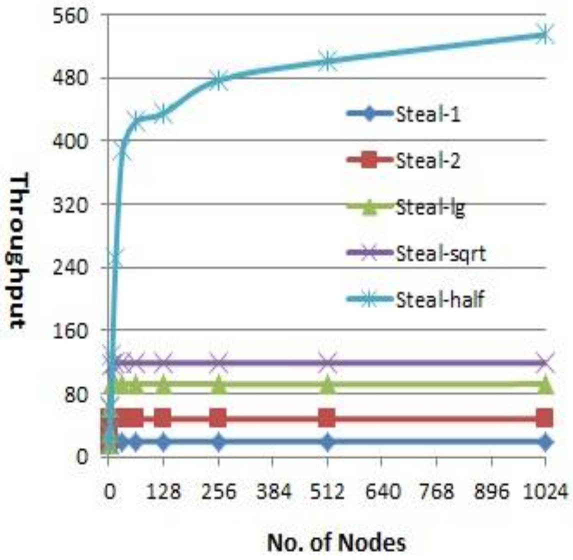


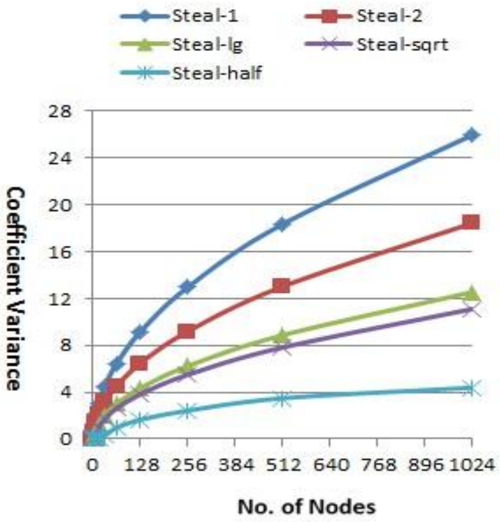
Results and Discussions

- 1. Correction Validation
- Run small experiments to trace the procedure of work stealing
- For large experiments, we see that the load balancing is good: the coefficient variance is close to zero

2. Optimal Parameters of Work Stealing

- Amount of jobs to steal
- average job length = 0.5 seconds, pollInterval = 0.05 seconds, numCoresPerNode = 8, totalNumJobs = 10000000, numNeighbors = 2



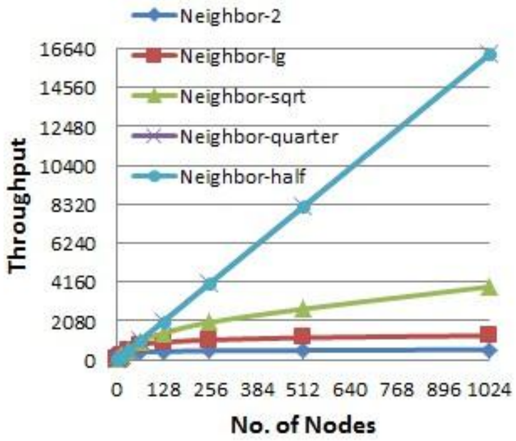


Change of throughput with respect to the number Of nodes for different steal policy

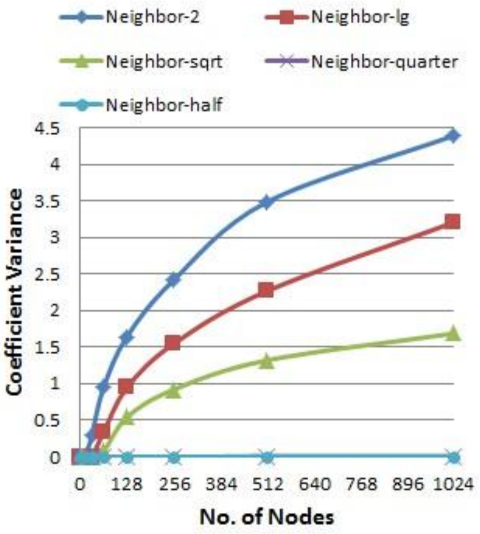
Change of coefficient variance with respect to number of nodes for different steal policy

No. of Neighbors a node has

- average job length = 0.5 seconds, pollInterval = 0.05 seconds, numCoresPerNode = 8, totalNumJobs = 10000000, steal-half policy



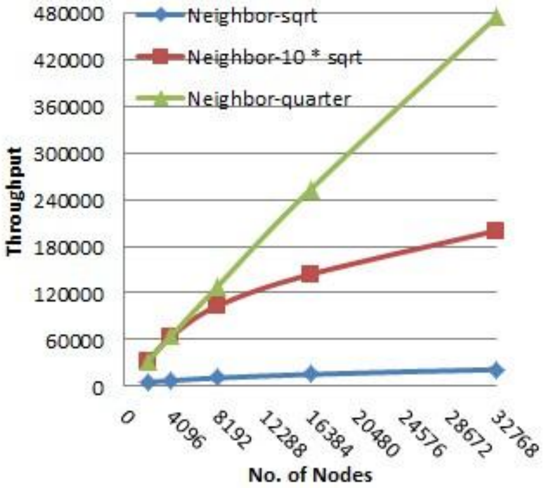
Change of throughput with respect to number of nodes for different number of neighbors



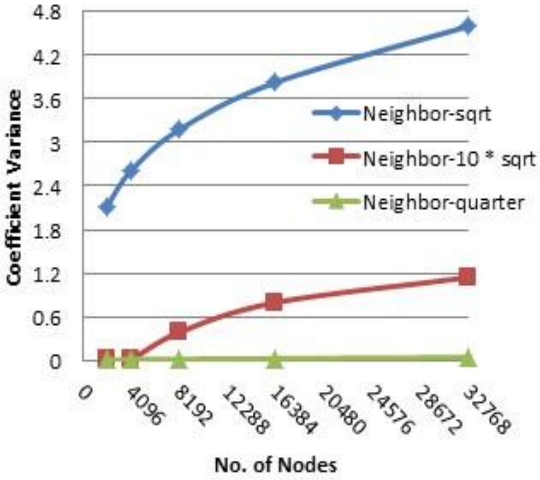
Change of coefficient variance with respect to number of nodes for different number of neighbors

No. of Neighbors a node has

- A quarter neighbors is too much in reality



Change of throughput with respect to number of nodes for different number of neighbors



Change of coefficient variance with respect to number of nodes for different number of neighbors

Poll Interval

- A node steals jobs from its neighbors, but all of which have no jobs. The node waits for some time and then tries to steal jobs again.
- Intuitively, the longer the average job length is, the larger the interval should be.
- numCoresPerNode = 8, totalNumJobs = 100000000, numNeighbors = a quarter of number of all nodes and steal-half policy.
- Results of changing the poll interval

	Poll Interval = 0.01			Poll Interval = 0.1			Poll Interval = 1			Poll Interval = 10			Poll Interval = 100		
numNode	Throu	coVar	Time(s)	Throu	coVar	Time(s)	Throu	coVar	Time(s)	Throu	coVar	Time(s)	Throu	coVar	Time(s)
1	0.0016	0.0	25.597	0.0016	0.0	21.439	0.0016	0.0	22.555	0.0016	0.0	21.857	0.0016	0.0	22.871
2	0.0032	2.158E-5	28.986	0.0032	1.754E-4	26.598	0.0032	3.526E-5	26.384	0.0032	8.604E-5	27.320	0.0032	4.932E-5	26.769
4	0.0064	1.412E-4	33.677	0.0064	9.711E-5	30.466	0.0064	3.842E-5	28.245	0.0064	6.960E-5	28.298	0.0064	9.040E-5	29.264
8	0.0128	1.391E-4	101.190	0.0128	2.397E-4	37.065	0.0128	1.729E-4	32.075	0.0128	1.573E-4	30.996	0.0128	1.167E-4	32.097
16	0.0256	2.625E-4	83.181	0.0256	2.321E-4	38.224	0.0256	1.956E-4	32.472	0.0256	2.000E-4	31.961	0.0256	3.013E-4	34.292
32	0.0512	2.910E-4	158.198	0.0512	3.230E-4	46.680	0.0512	3.636E-4	36.514	0.0512	2.925E-4	35.394	0.0512	4.164E-4	35.075
64	0.1024	4.359E-4	400.326	0.1024	4.105E-4	78.006	0.1024	4.518E-4	47.806	0.1024	4.435E-4	43.423	0.1024	4.942E-4	42.486
128	0.2048	6.968E-4	1059.367	0.2048	6.340E-4	157.946	0.2048	6.993E-4	62.401	0.2048	6.292E-4	50.774	0.2048	5.969E-4	49.617
256	0.4095	8.412E-4	4084.424	0.4095	9.090E-4	390.697	0.4096	9.440E-4	89.999	0.4096	8.843E-4	60.548	0.4096	9.072E-4	54.728
512	0.8192	1.273E-3	13612.788	0.8191	1.275E-3	1721.174	0.8191	1.260E-3	183.849	0.8190	1.299E-3	75.227	0.8192	1.294E-3	61.949
1024	1.6380	1.829E-3	67102.279	1.6382	1.821E-3	5643.013	1.6381	1.926E-3	666.370	1.6381	1.836E-3	127.173	1.6383	1.794E-3	79.552

Number of Cores a node has

- A node could have thousands of cores in the future
- are average job length = 5000 seconds, totalNumJobs = 100000000, numNeighbors = a quarter of number of all nodes, poll interval = 100 seconds and steal-half policy.

- Results of changing the number of cores of a node

	numCoresPerNode = 8			numCoresPerNode = 100			numCoresPerNode = 500			numCoresPerNode = 1000		
numNode	Throu	CoVari	RealTime(S)	Throu	CoVari	RealTime(S)	Throu	CoVari	RealTime(S)	Throu	CoVari	RealTime(S)
1	0.0016	0.0	22.871	0.0200	0.0	32.297	0.1000	0.0	41.378	0.2000	0.0	48.739
2	0.0032	4.932E-5	26.769	0.0400	6.266E-5	33.758	0.2000	2.048E-5	51.532	0.4000	4.830E-5	55.485
4	0.0064	9.040E-5	29.264	0.0800	1.451E-4	41.583	0.4000	8.872E-5	54.924	0.7999	6.235E-5	59.988
8	0.0128	1.167E-4	32.097	0.1600	1.295E-4	46.108	0.7999	1.199E-4	62.654	1.6000	1.498E-4	66.682
16	0.0256	3.013E-4	34.292	0.3200	2.926E-4	52.426	1.5999	2.490E-4	66.478	3.1999	1.939E-4	80.340
32	0.0512	4.164E-4	35.075	0.6400	3.252E-4	56.147	3.1991	3.194E-4	88.232	6.3979	3.942E-4	108.869
64	0.1024	4.942E-4	42.486	1.2798	4.816E-4	63.084	6.3967	4.394E-4	99.762	12.788	4.797E-4	125.868
128	0.2048	5.969E-4	49.617	2.5593	6.896E-4	75.425	12.7889	6.560E-4	154.008	25.555	6.873E-4	156.541
256	0.4096	9.072E-4	54.728	5.1176	8.907E-4	95.916	25.5491	1.019E-3	168.375	51.0060	1.203E-3	256.397
512	0.8192	1.294E-3	61.949	10.231	1.257E-3	137.949	50.9885	1.662E-3	243.435	101.6017	2.214E-3	369.082
1024	1.6383	1.794E-3	79.552	20.446	1.917E-3	196.736	101.5792	2.696E-3	480.852	201.5790	4.285E-3	906.012

Performance Reulsts

- average job length of 5000 seconds and use the optimal combination of parameters, that is steal-half policy, number of neighbors is a quarter of number of all nodes, poll interval is 100 seconds.
- Group one: 10 billion jobs and each node has 8 cores and we double the number of nodes every time
- Group two: each node has 1000 cores and we double the number of nodes every time and set the number of jobs 10 times of the number of all cores.

Results of Group One

No. of Nodes	Throughput	Coefficient Variance	Real Time(s)
1	0.001600003	0	2315.186
2	0.003200005	5.58E-06	2755.472
4	0.006399961	5.07E-06	2908.035
8	0.012799972	1.36E-05	3207.9
16	0.0255998	2.78E-05	3358.33
32	0.051199758	3.80E-05	3707.142
64	0.10239979	4.66E-05	4932.786
128	0.204800523	6.64E-05	5637.825
256	0.409599812	8.55E-05	6214.509
512	0.819197885	1.24E-04	6671.319
1024	1.638387861	1.85E-04	8606.809
2048	3.276828023	2.62E-04	11754.683
4096	6.553535084	3.66E-04	13668.893
8192	13.10698003	5.21E-04	18606.877
16384	26.2135383	7.35E-04	25383.456
32768	52.42464599	1.05E-03	39392.475
65536	104.8427879	1.48E-03	98527.071
131072	209.6592753	2.08E-03	538956.397
262144			
524288			
1048576			

Results of Group Two

No. of Nodes	Throughput	Coefficient Variance	Real Time(s)
1	0.178624945	0	0.992
2	0.353243131	0.001	1.404
4	0.70786606	0.009572617	1.128
8	1.410581173	0.005118594	1.297
16	2.81002987	0.006007183	1.785
32	5.62917367	0.006420572	2.672
64	11.20161569	0.025181497	4.701
128	22.36084491	0.023579927	9.339
256	44.54836015	0.029038926	22.324
512	88.7077843	0.036974344	68.622
1024	177.0042483	0.039759638	188.872
2048	353.452133	0.040117491	747.514
4096	705.0215427	0.042431897	6864.872
8192	1410.800987	0.04194238	28637.461
16384	2819.51508	0.042909697	126936.902
32768	/	/	/
65536	/	/	/
131072	/	/	/

Comparison Between two simulators

- The scalability of the centralized one is not as good as the distributed one. The centralized server is a bottleneck. The upper bound of throughput is around 1000. The program runs very fast, it takes about 20 hours to run exascale experiments.
- The distributed simulator scales very well, the increase of throughput is linear with that of number of nodes. As there are so many events in the system, it takes longer to run experiments at the same scale as the centralized one

Conclusion and Future work

- Both the simulators could run experiments at exascale, though it takes longer for the distributed simulator.
- The distributed simulator beats the centralized one in terms of scalability and reliability
- Future work involves memory issues and playing with parameters, such as poll interval, to reduce the real time for distributed simulator. Maybe a fully distributed simulator is our next goal.