# Scheduling Many-Task Applications on Multi-Clouds and Hybrid Clouds

Shifat P. Mithila, Peter Franz, Gerald Baumgartner

# Who am I?

- Undergraduate research assistant
- Upgraded the platform to multiple cloud sites
- Learned algorithms through working with the code and explanations

# Status Quo Multi-Computer Approaches

| Approach | Performance | Networking | Resources | Parallelism |
|---|---|---|---|---|
| Supercomputers | Predictable | Low Latency + Robust | Homogeneous | Fine Grained Parallelism |
| Desktop Grids | Unpredictable | High Latency + Unreliable | Heterogeneous | No parallelism; Independent Tasks |
| Cloud Virtual Resources | Unpredictable | Moderate Latency + Relialiable | Heterogeneous | Coarse Parallelism; Map Reduce |

# Virtual Machine Challenges

- Performance information is unpredictable before resources are provisioned
- Latency to neighbors varies as host pcs change
- Heterogeneous resources
- Performance variability - competition for host os resources

# Supercomputer

- Forehand performance knowledge
- Robust and low latency networking
- Expensive
- Fine grained parallelism

# Large-scale desktop-based master-worker grids

- Heterogenous and variable performance of nodes
- Unreliable and high latency networking
- Donated CPU cycles
- Little to no parallelism

# Cloud Computing

- Virtual Machines
- Heterogenous and variable performance of nodes
- Robust but moderate latency networking
- Pay per use
- <u>Fine grained parallelism (?)</u>

# Why you should be interested?

- Cheaper
- Higher availability
- Provider agnostic; allows price shopping
- Utilize your existing hardware
- Scale resources depending on task

A task based decentralized-vector-scheduling platform, minimizing propagation delay with latency based clustering.

# Task Based

- Tasks break large problems into smaller steps

- Some tasks require other tasks to be completed to start

- Tasks are python script to run with passed parameters

$$
\begin{aligned}
&R^{p1p2}_{h1h2} = v\_vvoo^{p1p2}_{h1h2} - t\_vo^{p1}_{h7} * (v\_ovoo^{h7p2}_{h1h2} - 0.5 * t\_vo^{p2}_{h8} * \\
&(v\_oooo^{h7h8}_{h1h2} + (v\_ooov^{h7h8}_{h1p3} - 0.5 * v\_oovv^{h7h8}_{p3p4} * t\_vo^{p4}_{h1}) * t\_vo^{p3}_{h2} + \\
&0.5 * t\_vvoo^{p5p6}_{h1h2} * v\_oovv^{h7h8}_{p5p6}) + t\_vo^{p3}_{h2} * (v\_ovov^{h7p2}_{h1p3} - 0.5 * \\
&t\_vo^{p4}_{h1} * v\_ovvv^{h7p2}_{p3p4}) - (f\_ov^{h7}_{p3} - t\_vo^{p4}_{h4} * v\_oovv^{h4h7}_{p3p4}) * t\_vvoo^{p2p3}_{h1h2} - \\
&t\_vvoo^{p2p7}_{h2h4} * (v\_ooov^{h4h7}_{h1p7} + t\_vo^{p3}_{h1} * v\_oovv^{h4h7}_{p3p7}) + 0.5 * t\_vvoo^{p3p4}_{h1h2} * \\
&v\_ovvv^{h7p2}_{p3p4}) + t\_vo^{p3}_{h2} * (v\_vvov^{p1p2}_{h1p3} - 0.5 * t\_vo^{p4}_{h1} * v\_vvvv^{p1p2}_{p3p4}) + \\
&(f\_oo^{h6}_{h1} + (f\_ov^{h6}_{p6} + t\_vo^{p4}_{h4} * v\_oovv^{h4h6}_{p4p6}) * t\_vo^{p6}_{h1} - t\_vo^{p4}_{h4} * \\
&v\_ooov^{h4h6}_{h1p4} - 0.5 * t\_vvoo^{p4p5}_{h1h5} * v\_oovv^{h5h6}_{p4p5}) * t\_vvoo^{p1p2}_{h2h6} + (f\_vv^{p2}_{p3} - \\
&t\_vo^{p4}_{h4} * v\_ovvv^{h4p2}_{p3p4} - 0.5 * t\_vvoo^{p2p4}_{h4h5} * v\_oovv^{h4h5}_{p3p4}) * t\_vvoo^{p1p3}_{h1h2} + \\
&0.5 * t\_vvoo^{p1p2}_{h6h8} * (v\_oooo^{h6h8}_{h1h2} + t\_vo^{p6}_{h2} * (v\_ooov^{h6h8}_{h1p6} + 0.5 * \\
&t\_vo^{p4}_{h1} * v\_oovv^{h6h8}_{p4p6}) - 0.5 * t\_vvoo^{p3p4}_{h1h2} * v\_oovv^{h6h8}_{p3p4}) + t\_vvoo^{p1p3}_{h2h3} * \\
&(v\_ovov^{h3p2}_{h1p3} - t\_vo^{p5}_{h1} * v\_ovvv^{h3p2}_{p3p5} - 0.5 * t\_vvoo^{p2p5}_{h1h5} * v\_oovv^{h3h5}_{p3p5}) + \\
&0.5 * t\_vvoo^{p3p4}_{h1h2} * v\_vvvv^{p1p2}_{p3p4}
\end{aligned}
$$

Spin-orbital coupled-cluster singles-and-double doubles equation

# Task Passing

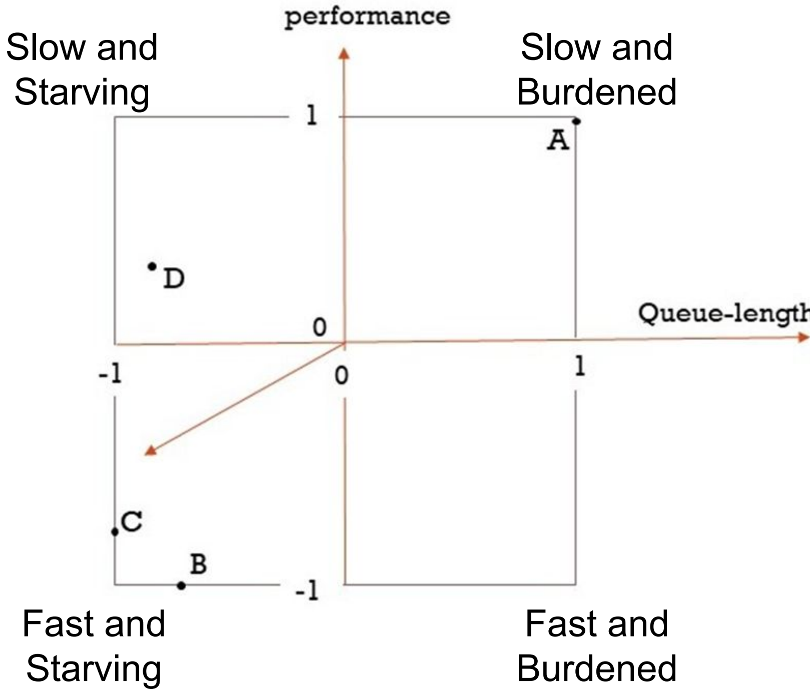
- All nodes must be utilized
- Nodes must share
- Must be a fast decision
- Balancing act between multiple considerations

# Decentralized

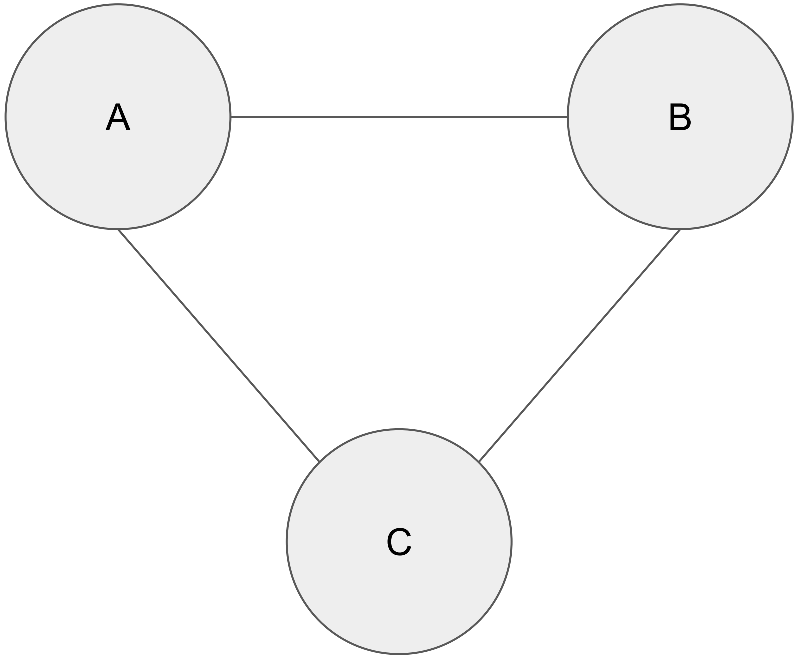
- Avoiding a centralized bottleneck
- Short-running parallel-tasks on network connected heterogeneous resources
  - Controller is remote from nodes in cloud environment
  - Short running means less time for decisions to come from the controller
  - Controller needs to manage large amounts performance information
- Need task passing decision to be made at node level

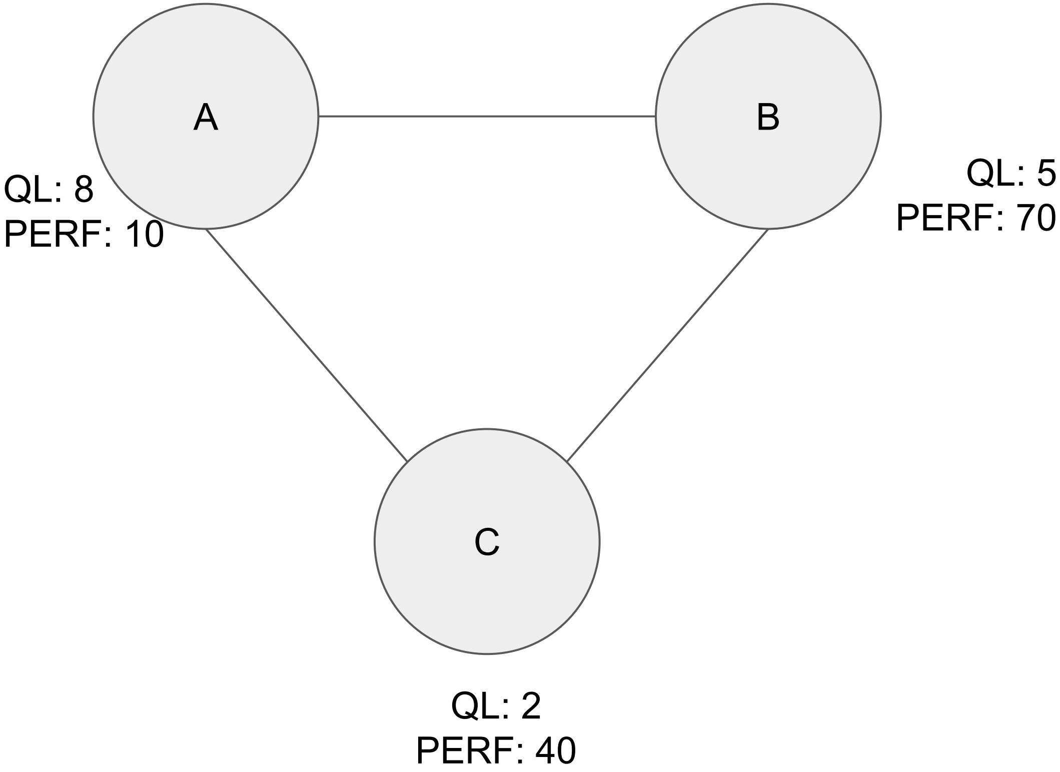
# Vector-Scheduling

- Lightweight decentralized algorithm
- Nodes connected in overlay graph
- Nodes send performance information to neighbors
- Measurements are normalized
- Using an experiment defined flow vector tasks sent in desired direction to neighbors



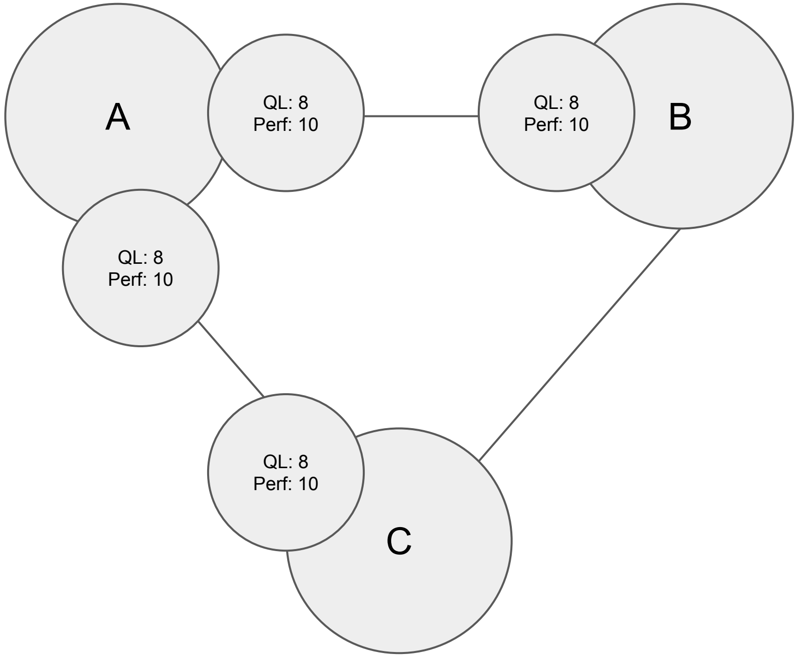
# Vector Scheduling worked example





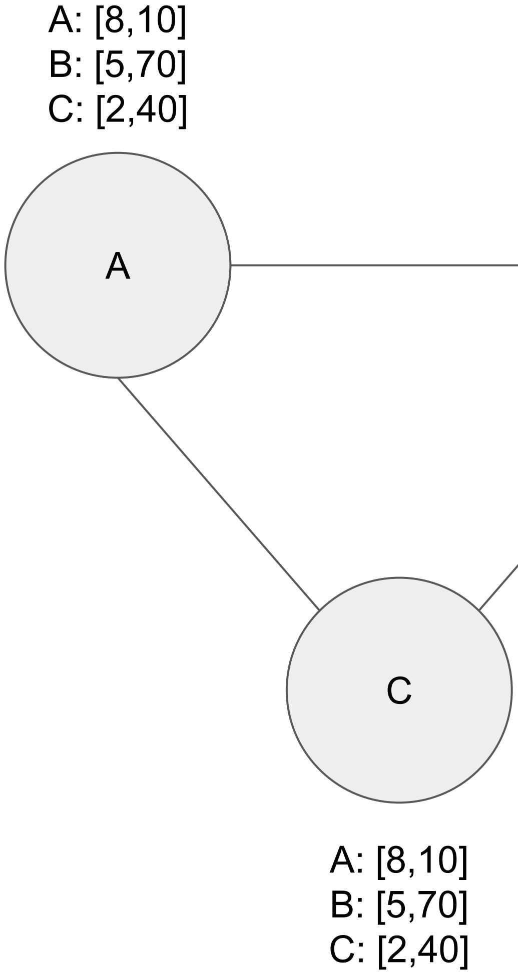

Nodes probe a characteristic

```
Ql  = self.queueLen()
Perf = self.runBenchmark()
```



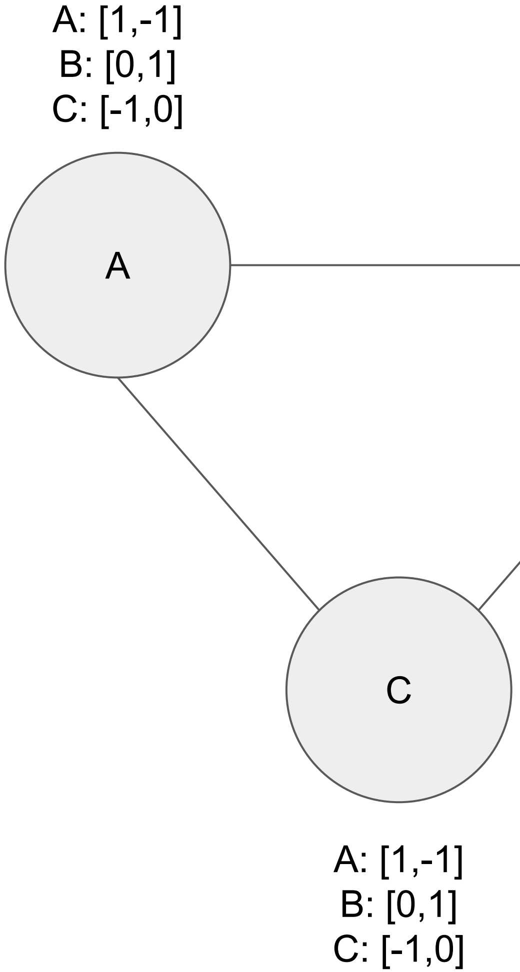

## Nodes provide information to neighbors

```
For each neighbor:
        shipQLToNeighbor(neighbor)
        shipPerfToNeighbor(neighbor)
```



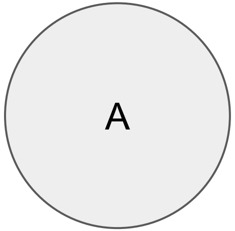

Nodes keep track of the components in a vector for each neighbor [QueueLength, Performance]

```
For each neighbor:
        neighborVector =
        [
          reciveQL(neighbor),
          receivePerf(neighbor),
        ]
```




Nodes normalize vectors

# Scoring with a flow-vector




Flow Vector:
[-1,-0.25]

| Node | Queue Length | Performance | Score |
|---|---|---|---|
| A | 1 | -1 | -0.75 |
| B | 0 | 1 | -0.25 |
| C | -1 | 0 | 1 |

A: (1 * -1) + (-1 * -0.25) = -0.75

B: (0 * -1) + (1 * -0.25) = -0.25

C: (-1 * -1) + (0 * -0.25) = 1

# Thrashing

- Tasks passed back and forth with no completion
- Want passing, but not for minor gains
- “Statis vector” to minimize thrashing

# Stasis-Vector

A

Flow Vector:
[-1,-0.25]

Stasis Vector:
[0,-0.5]

| Node | Queue Length | Performance | Score |
|---|---|---|---|
| A | 1 | -1 | ~~-0.75~~ -0.625 |
| B | 0 | 1 | -0.25 |
| C | -1 | 0 | 1 |

A: ( [1+0] * -1) + ([-1 + -0.5] * -0.25) = -0.625

# Adding latency

- Vector scheduling is an extremely expandable
- Add Latency to scheduling
- Allows us to consider nodes relative locations

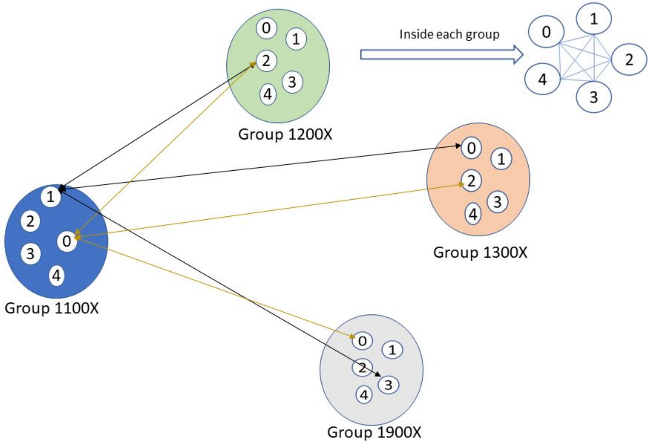
# Network Connectivity

- Full connectivity between all nodes has poor performance
- High latency connections should be avoided
- More neighbors increases overhead of vector scheduling

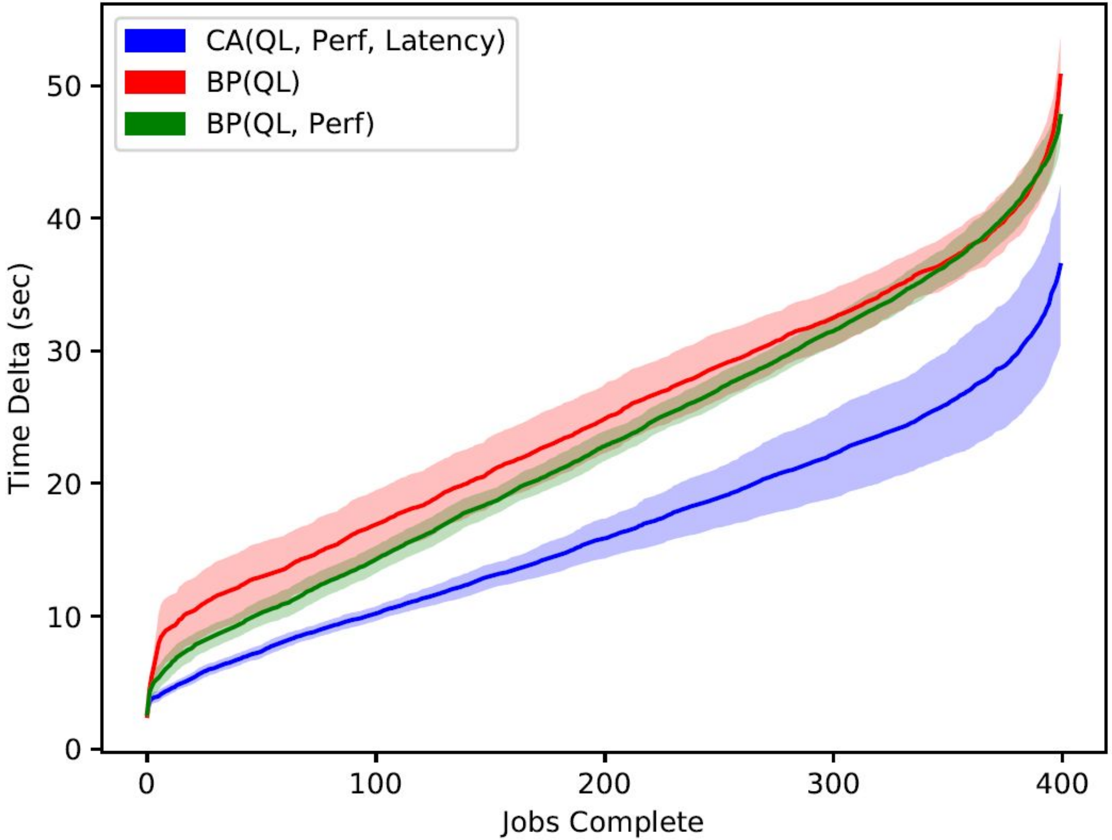
# Centralized Clustering Algorithm

- Measure latency to neighbors
- Nodes form a full graph with closest neighbors, Clique
- Keep some connections to distant groups

- Reduces number of neighbors to evaluate
- Removes slower connections
- Maintains connectivity



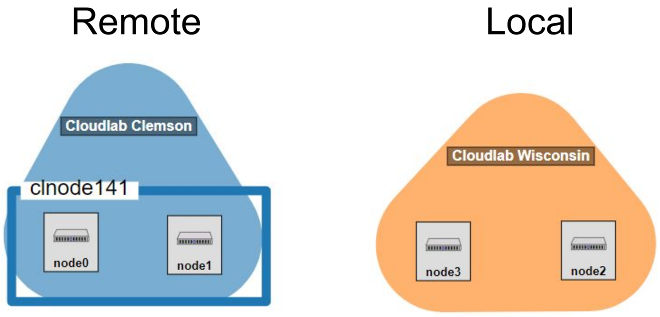
# Latency + Clustering increases performance



# Hybrid + Multi-Site Resource Proof of Concept

- 400 65 x 65 matrix multiplication tasks, with 50KB payload file included
- Demonstrates our capacity to stitch together multiple resource types and sites
- Limitation with handling LSU firewall, more development needed

| experiment | resources | initial vector | final vector | time |
|---|---|---|---|---|
| Local | 2 PCs | [-1, -0.5, 0.5] | [-1, 0, -0.25] | 457.95 |
| Hybrid | 2 PCs, 2 VMS | [-1, -0.5, 0.5] | [-1, 0, -0.25] | 280.47 |



# Extending Research with Heterogeneity

- 400 65 x 65 matrix multiplication tasks, with 50KB payload file
- Heterogeneous performance
- Random graph construction, 20%


| Site | Hardware Type | Workers | CPU | CPU Speed MHZ |
|---|---|---|---|---|
| Clemson | VM | 4 | 2 cores per | - |
| Clemson | c4130 | 10 | Intel E5-2680v3 | 2500 |
| Utah | d6515 | 10 | AMD EPYC 7452 | 2400 |
| Utah | m510 | 12 | Intel Xeon D-1548 | 2000 |
| Wisconsin | c220g2 | 10 | Haswell | 2600 |

# 3D Vector Scheduling in Multi-Site Cloud

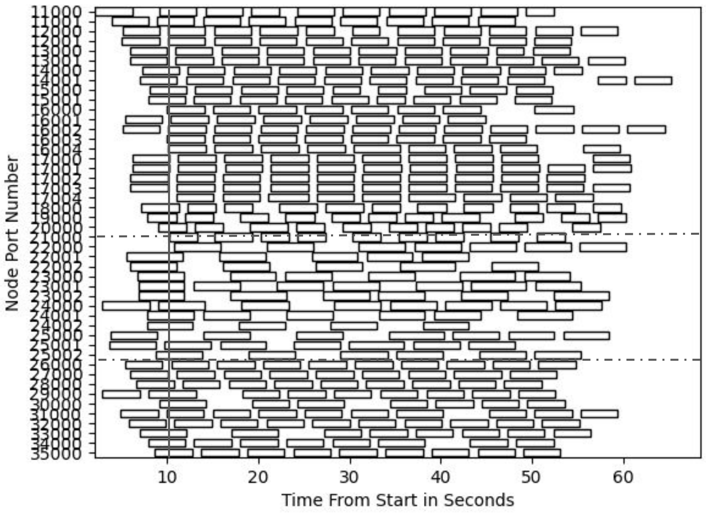
- Varying vector configurations
- Ignoring latency had worst performance
- Avoiding latency is good for the middle and end
- Intentionally pushing tasks at the start, leads to good task saturation

Table 1: Performance comparison for different [queue, performance, latency] scheduling vectors with vector swap after 30 seconds.

| Experiment | initial vector | final vector | time |
|---|---|---|---|
| Ignore latency | [-1, -0.5, 0] | [-1, 0.5, 0] | 78.9 |
| Emphasize queue | [-1, 0, 0] | [-0.5, -0.5, -0.25] | 72.9 |
| Emphasize performance | [-1, -0.5, 0.5] | [-1, 0, -0.25] | 67.6 |

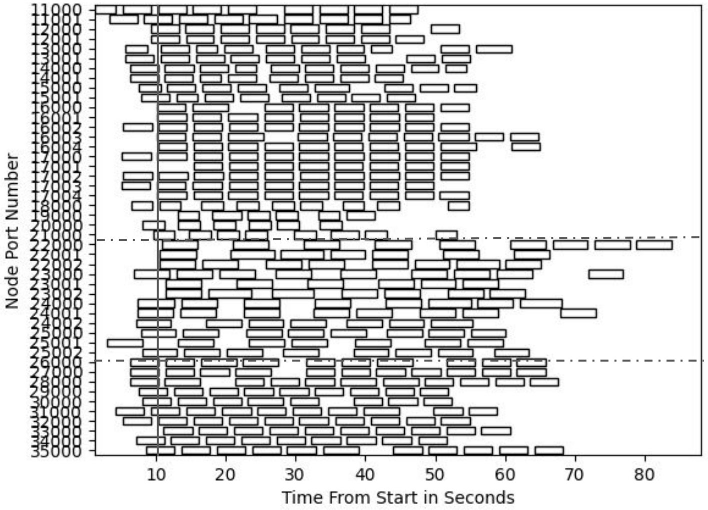
# Reading Tea Leaves

Use Latency + Push Away to Start




- Using latency to start has 93% Node utilization after 10 seconds
- All sites have work until the end

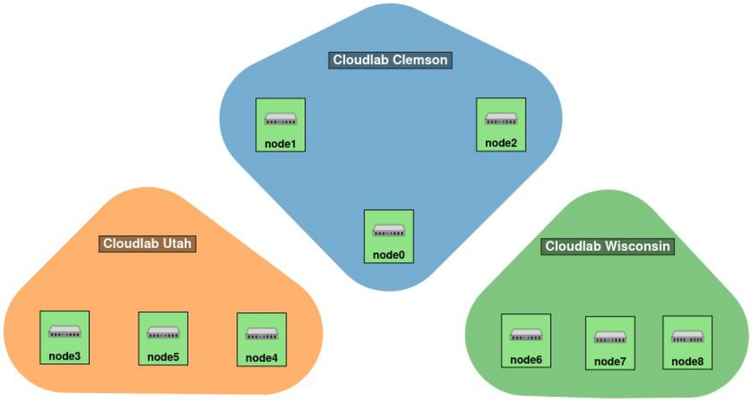
Ignore Latency




- Ignoring latency has 73% Node utilization after 10 seconds
- Some sites starve at the end

# Extending Research with Heterogeneity

- 400 65 x 65 matrix multiplication tasks, with 50KB payload file
- Heterogeneous performance
- Random graph construction, 20%

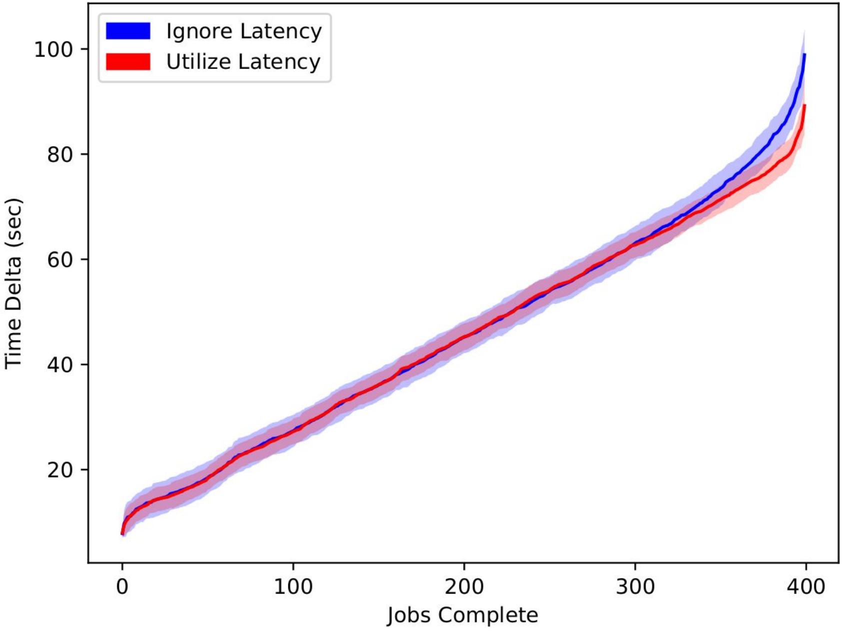



| Site | Hardware Type | Workers | CPU | CPU Speed MHZ |
|---|---|---|---|---|
| Clemson | c8220 | 10 | Intel E5-2660v 2 | 2200 |
| Utah | m510 | 16 | Intel Xeon D-1548 | 2000 |
| Wisconsin | c220g1 | 20 | Haswell | 2400 |

# 3D Vector Scheduling in Multi-Site Cloud

- Varying vector configurations
- Ignoring latency had worst performance
- Intentionally pushing tasks at the start, leads to good task saturation

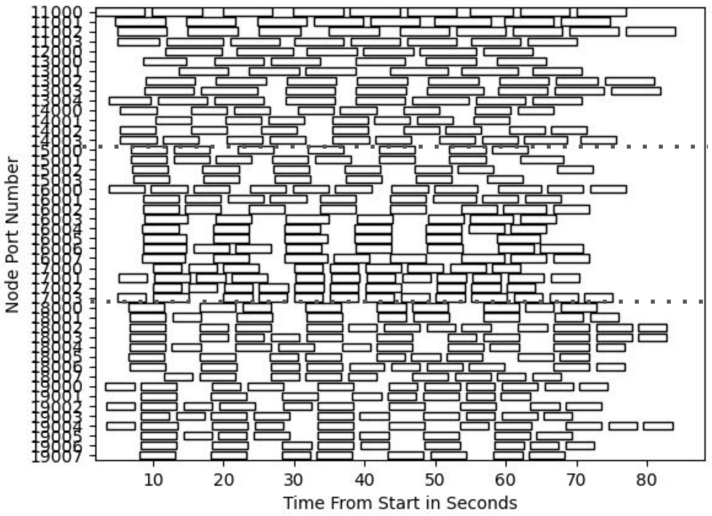
| Name | Initial vector | Second Vector | Average Time |
|---|---|---|---|
| Ignore Latency | -1,-0.5,0 | -1,0.5,0 | 100.93 |
| Minimize Latency | -1,-0.25,-0.25 | -1,0,-0.25 | 91.55 |
| Push Tasks | -1,-0.5,0.5 | -1,0,-0.25 | 89.82 |



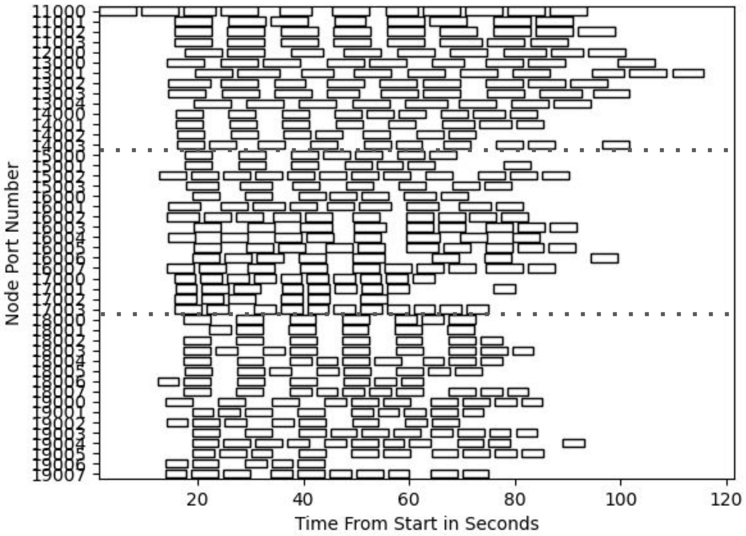
Ignore Latency
Utilize Latency
Time Delta (sec)
100
80
60
40
20
0
100
200
300
400
Jobs Complete

# Reading Tea Leaves

Use Latency + Push Away to Start




Ignore Latency

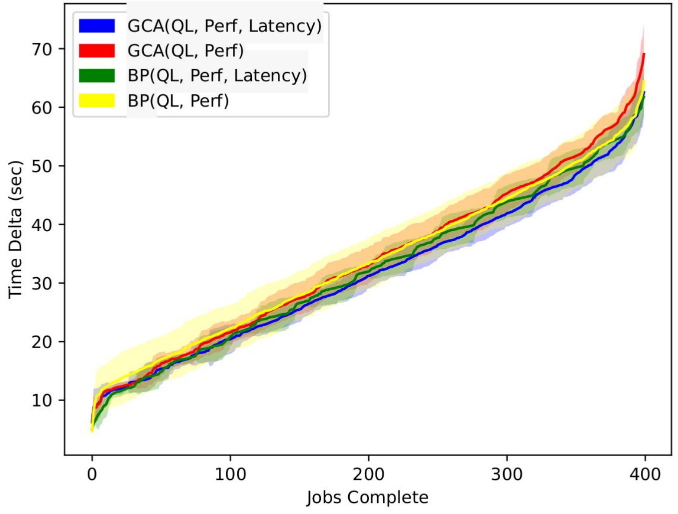



- Nodes quickly utilized
- All sites have work until the end

- Slow to utilize all nodes
- Some sites starve at the end

# Clustering Algorithm Limitations

- 46 worker nodes, 400 65x65 Matrix Mult tasks
    - GCA = Graph Construction Algorithm, Cluster
    - BP = Brain Peterson Algorithm, 20% Connectivity
- Clustering algorithm on real world clusters have oversize cliques
- Limited trials



# Further Work

- Improve adaptation of clustering algorithm parameters
- Trial automation

- General algorithm for changing flow vectors at runtime
- Probe hardware and system information to include as a vector component
- Task dependencies
- Tasks ran on collaboration of multiple nodes