# Network and Load Aware Resource Allocator for Parallel Programs

Ashish Kumar  Naman Jain

December 14, 2019

# Introduction

Image source: *NASA, EU, CERN, InfraVec2*

# Introduction

Distributed-memory parallel programs

- More than one processing element using their own local memory during computation.

# Introduction

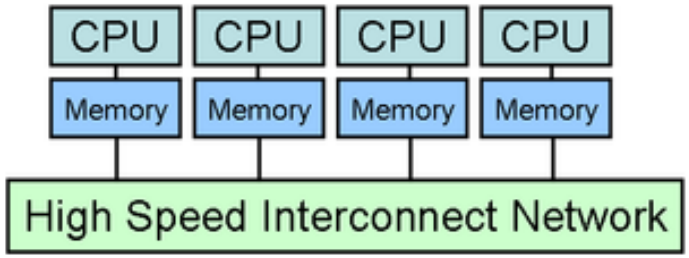
Distributed-memory parallel programs

- More than one processing element using their own local memory during computation.
- Processing elements can span multiple compute nodes.

# Introduction

Distributed-memory parallel programs

- More than one processing element using their own local memory during computation.
- Processing elements can span multiple compute nodes.
- Nodes connected to a communication network

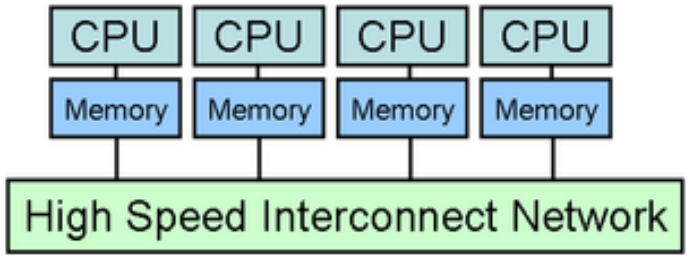



Source: *Wikipedia*

# Introduction

Distributed-memory parallel programs

- More than one processing element using their own local memory during computation.
- Processing elements can span multiple compute nodes.
- Nodes connected to a communication network



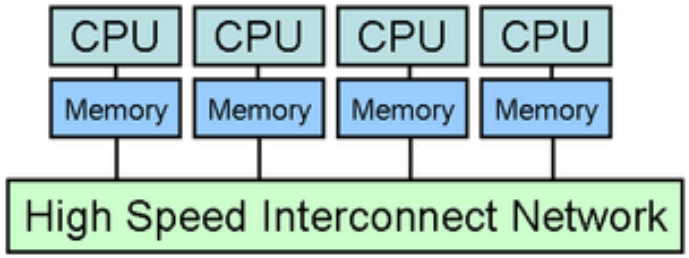

Source: *Wikipedia*

- Nodes work cooperatively to solve a single big problem

# Introduction

Distributed-memory parallel programs

- More than one processing element using their own local memory during computation.
- Processing elements can span multiple compute nodes.
- Nodes connected to a communication network



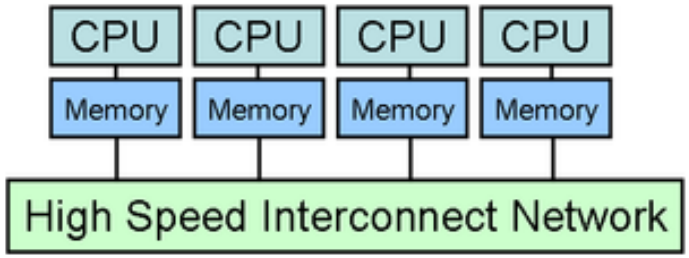

Source: *Wikipedia*

- Nodes work cooperatively to solve a single big problem
- Nodes exchange data through communications by sending and receiving messages.
- Uses Message Passing Interface (MPI) as "de facto" standard for message passing.

# Introduction

Distributed-memory parallel programs

- More than one processing element using their own local memory during computation.
- Processing elements can span multiple compute nodes.
- Nodes connected to a communication network




Source: *Wikipedia*

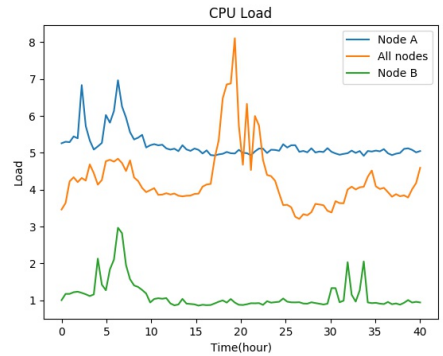
- Nodes work cooperatively to solve a single big problem
- Nodes exchange data through communications by sending and receiving messages.
- Uses Message Passing Interface (MPI) as "de facto" standard for message passing.
- Runs on cluster (shared or dedicated) or a supercomputer.

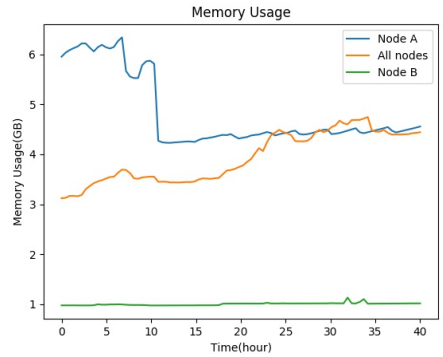
# Introduction

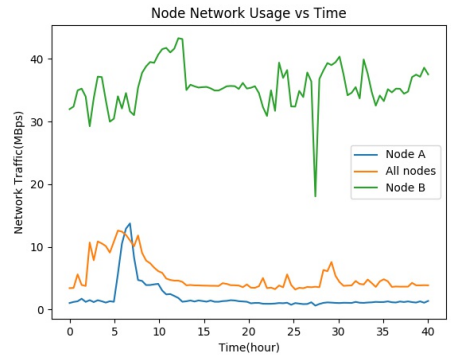
- Which nodes to run our job on?

# Introduction

- Which nodes to run our job on?
- Variations in utilization across nodes
- Variations in utilization across time

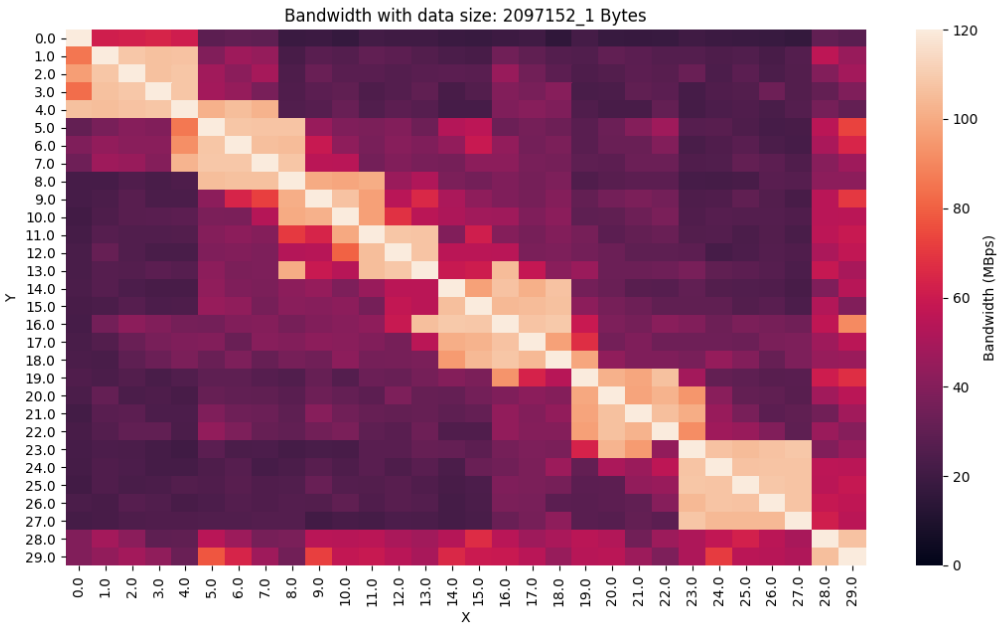


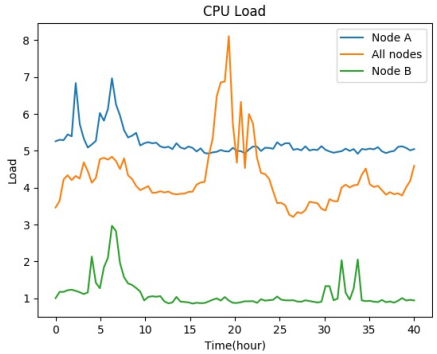



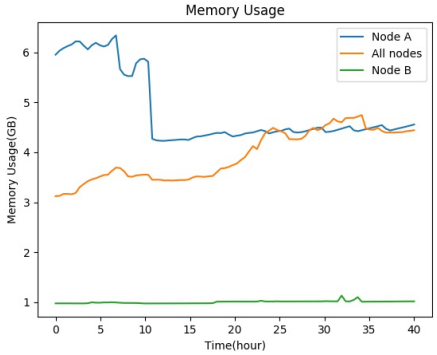



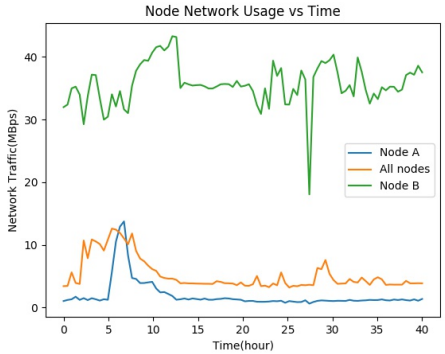
# Introduction

- Which nodes to run our job on?
- Variations in utilization across nodes
- Variations in utilization across time
- Network dynamics affects performance



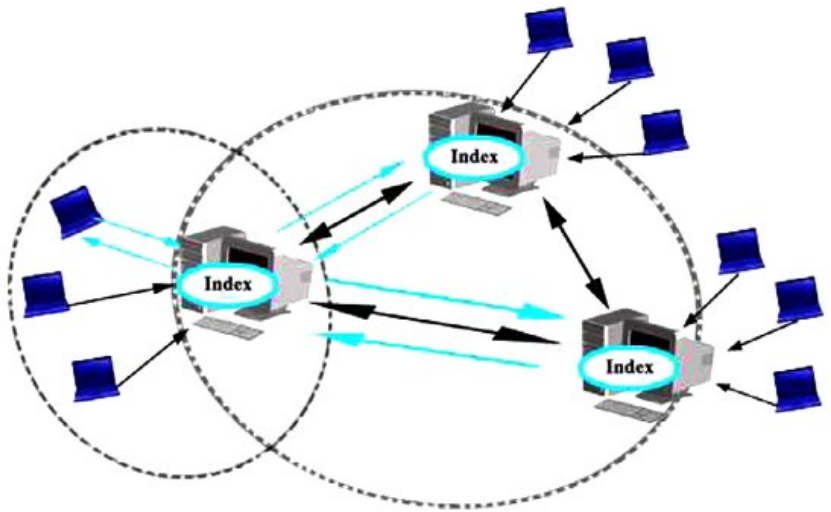



# Overview

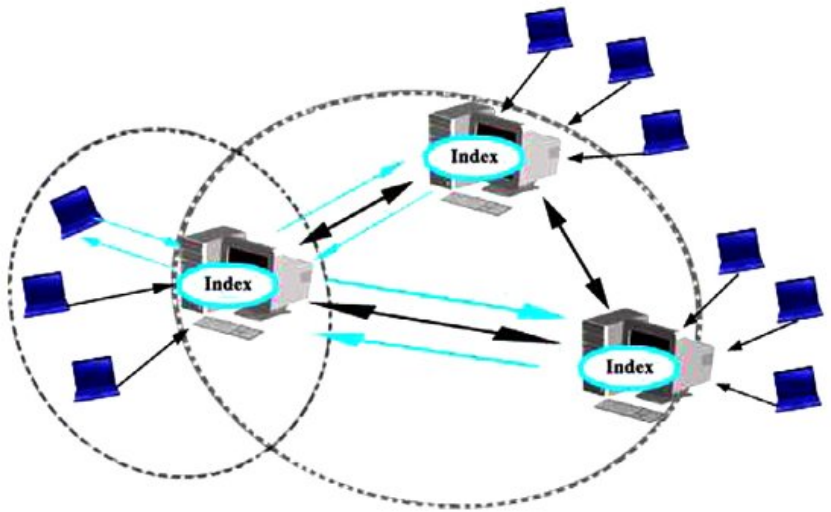
# Multi-Attribute Decision Making




Source: *Discovery of resources using MADM approaches for parallel and distributed computing*

# Multi-Attribute Decision Making

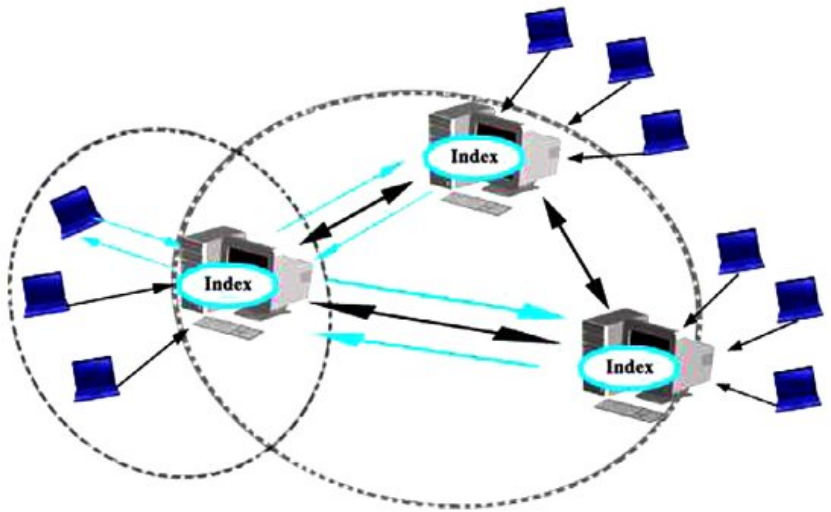
- Phase I:
  - Discover Resources
  - Simple Additive Weighting (SAW) method for shortlisting




Source: *Discovery of resources using MADM approaches for parallel and distributed computing*

# Multi-Attribute Decision Making

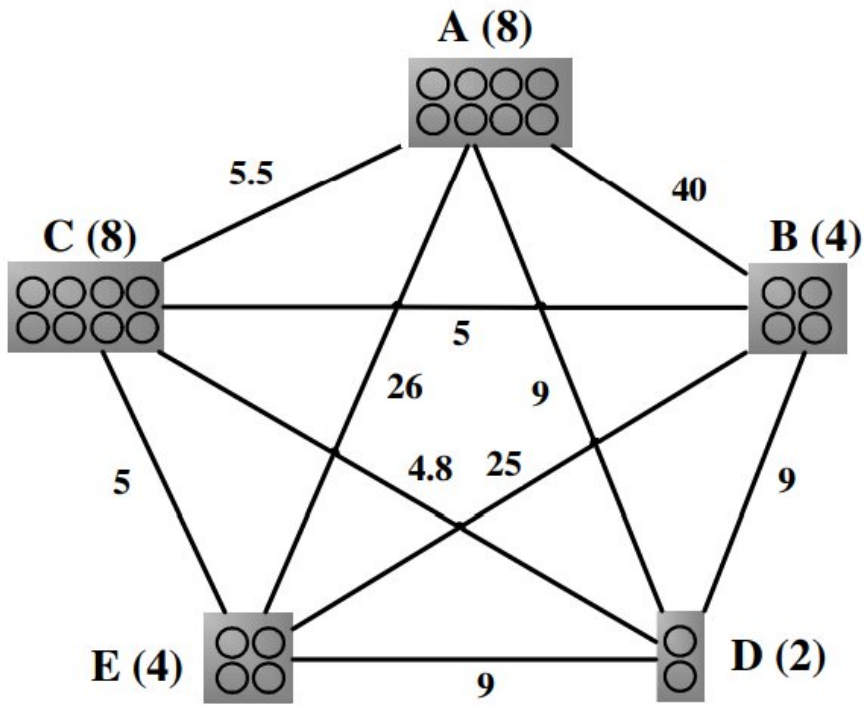
- Phase I:
  - Discover Resources
  - Simple Additive Weighting (SAW) method for shortlisting
- Phase II:
  - Integrated Multi-Attribute Decision Making (MADM) approach
  - Ranking based on pairwise comparisons




Source: *Discovery of resources using MADM approaches for parallel and distributed computing*

# Network-Bandwidth Aware Resource Broker

- Grid computing environment
- Multiple sites connected through high speed network




Source: *A Grid Resource Broker with Network Bandwidth-Aware Job Scheduling for Computational Grids*

# Network-Bandwidth Resource Broker

Average Total Power(ATP) of a site

$$ATP(S_i) = \left( \underbrace{\frac{\sum_{j=1}^{n} Pval_{ij} \times (1 - Pu_{ij})}{P(S_i)}}_{\text{Compute Power}} \times \alpha_{PE} + \underbrace{\frac{\sum_{j=1}^{n} Mval_{ij}}{P(S_i)}}_{\text{Memory Power}} \times (1 - \alpha_{PE}) \right) \times \alpha_{NE}$$

# Network-Bandwidth Resource Broker

Average Total Power(ATP) of a site

$$ATP(S_i) = \left( \underbrace{\frac{\sum_{j=1}^{n} Pval_{ij} \times (1 - Pu_{ij})}{P(S_i)}}_{\text{Compute Power}} \times \alpha_{PE} + \underbrace{\frac{\sum_{j=1}^{n} Mval_{ij}}{P(S_i)}}_{\text{Memory Power}} \times (1 - \alpha_{PE}) \right) \times \alpha_{NE}$$

In case, resources can't be find in 1 site.

$$\text{Aggregated Group Load} = \beta \times \sum E_{ij} + (1 - \beta) \times \sum ATP(S_i)$$

# Resource Allocation in Shared Cluster for Parallel Jobs

- Non exclusive access of nodes

# Resource Allocation in Shared Cluster for Parallel Jobs

- Non exclusive access of nodes
- Variation in load/utilization across time/nodes

# Resource Allocation in Shared Cluster for Parallel Jobs

- Non exclusive access of nodes
- Variation in load/utilization across time/nodes
- Topology does not capture the current state of network.

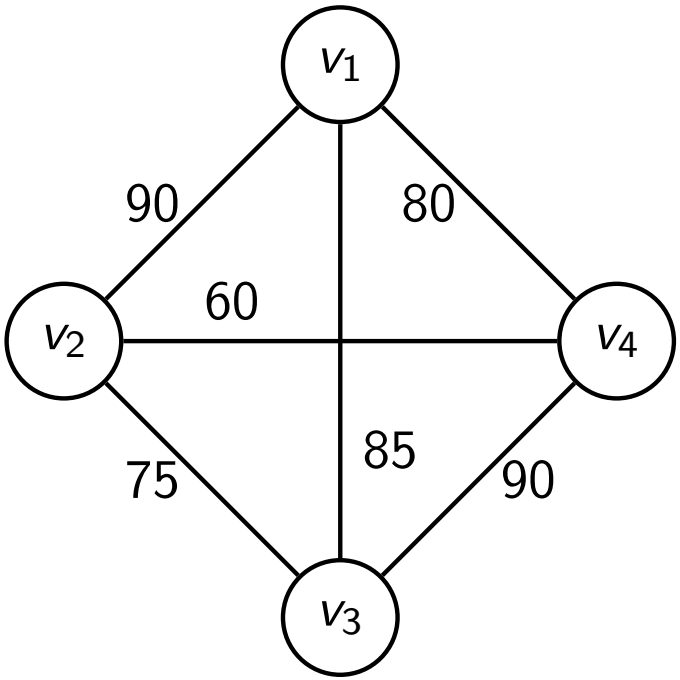
# Resource Allocation in Shared Cluster for Parallel Jobs

- Non exclusive access of nodes
- Variation in load/utilization across time/nodes
- Topology does not capture the current state of network.
- There may be contention and congestion in the network due to existing jobs.

# Resource Allocation in Shared Cluster for Parallel Jobs

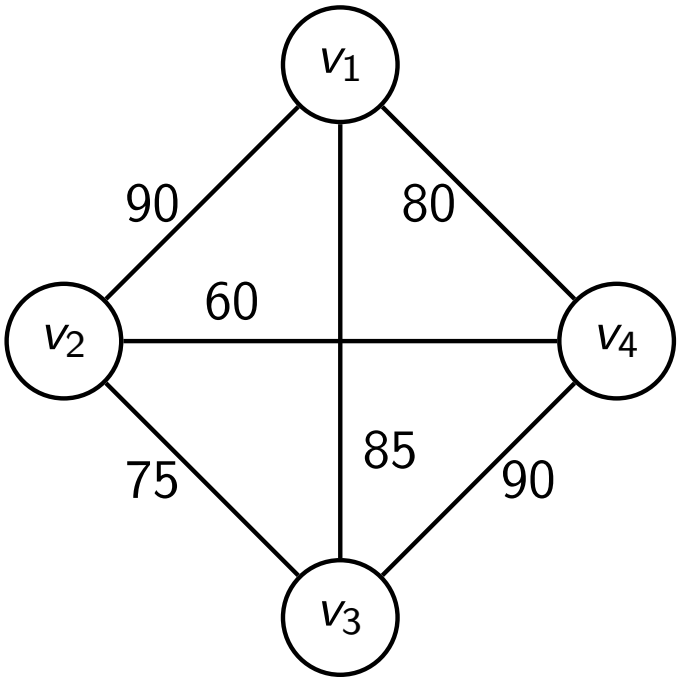
- Non exclusive access of nodes
- Variation in load/utilization across time/nodes
- Topology does not capture the current state of network.
- There may be contention and congestion in the network due to existing jobs.
- Some jobs may be communication-intensive,some may be compute-intensive.

# Allocation as Sub Graph Selection




- $\mathcal{G} = (\mathcal{V}, \mathcal{E})$

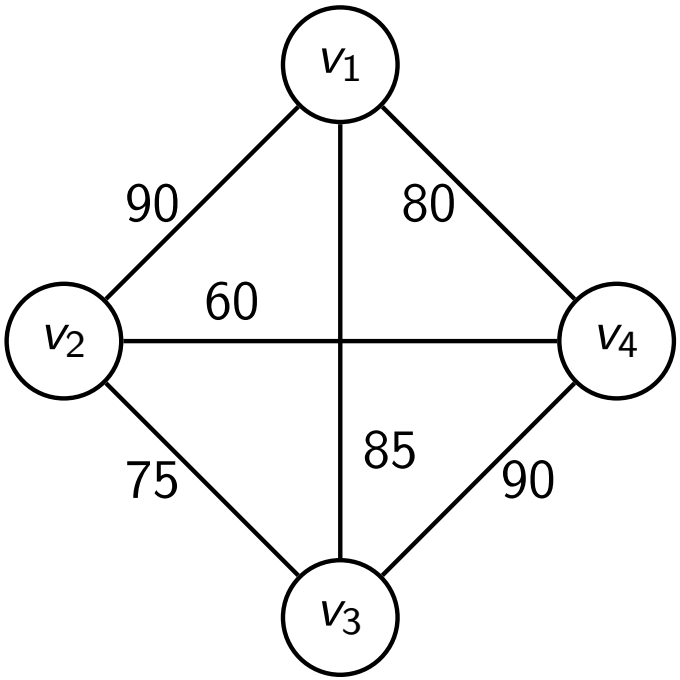
# Allocation as Sub Graph Selection




| Node | Compute load | #Cores |
|---|---|---|
| $v_1$ | 50.2 | 6 |
| $v_2$ | 43.5 | 8 |
| $v_3$ | 54.7 | 10 |
| $v_4$ | 38.3 | 4 |

- $\mathcal{G} = (\mathcal{V}, \mathcal{E})$
- Vertex $v \in \mathcal{V}$ : compute node having compute load $CL_v$ and available processor count $pc_v$

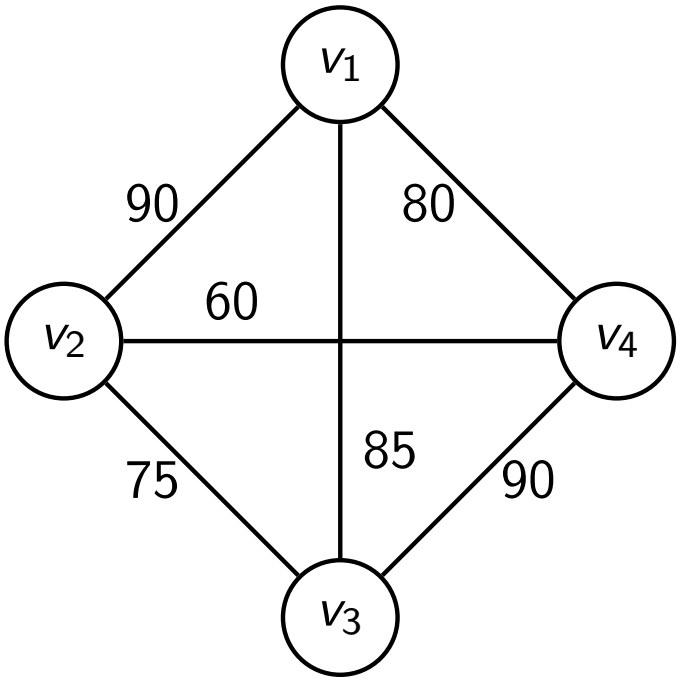
# Allocation as Sub Graph Selection




| Node | Compute load | #Cores |
|---|---|---|
| $v_1$ | 50.2 | 6 |
| $v_2$ | 43.5 | 8 |
| $v_3$ | 54.7 | 10 |
| $v_4$ | 38.3 | 4 |

- $\mathcal{G} = (\mathcal{V}, \mathcal{E})$
- Vertex $v \in \mathcal{V}$ : compute node having compute load $CL_v$ and available processor count $pc_v$
- Edge $e \in \mathcal{E}$ : network load $NL_{(u,v)}$ between compute nodes.

# Allocation as Sub Graph Selection




| Node | Compute load | #Cores |
|---|---|---|
| $v_1$ | 50.2 | 6 |
| $v_2$ | 43.5 | 8 |
| $v_3$ | 54.7 | 10 |
| $v_4$ | 38.3 | 4 |

- $\mathcal{G} = (\mathcal{V}, \mathcal{E})$
- Vertex $v \in \mathcal{V}$ : compute node having compute load $CL_v$ and available processor count $pc_v$
- Edge $e \in \mathcal{E}$ : network load $NL_{(u,v)}$ between compute nodes.
- $n$ : number of processes to be allocated
- Find a sub-graph such that the overall cost/load of the sub-graph is minimized and process demand is fulfilled.

# Some Definitions

- Compute load: measure of overall load on the node
  - Static attributes: clock speed, core count, total memory.
  - Dynamic attributes: CPU load, CPU utilization, available memory
  - Compute load, $\mathrm{CL}_v = \sum_{a \in \text{attributes}} w_a * val_{va}$

# Some Definitions

- Compute load: measure of overall load on the node
  - Static attributes: clock speed, core count, total memory.
  - Dynamic attributes: CPU load, CPU utilization, available memory
  - Compute load, $\mathrm{CL}_v = \sum_{a \in \text{attributes}} w_a * val_{va}$
- Network load: measure of load on the p2p network link
  - Latency
  - Bandwidth
  - Network load, $\mathrm{NL}_{(u,v)} = w_{lt}\mathrm{LT}_{(u,v)} + w_{bw}\mathrm{BW}_{(u,v)}$

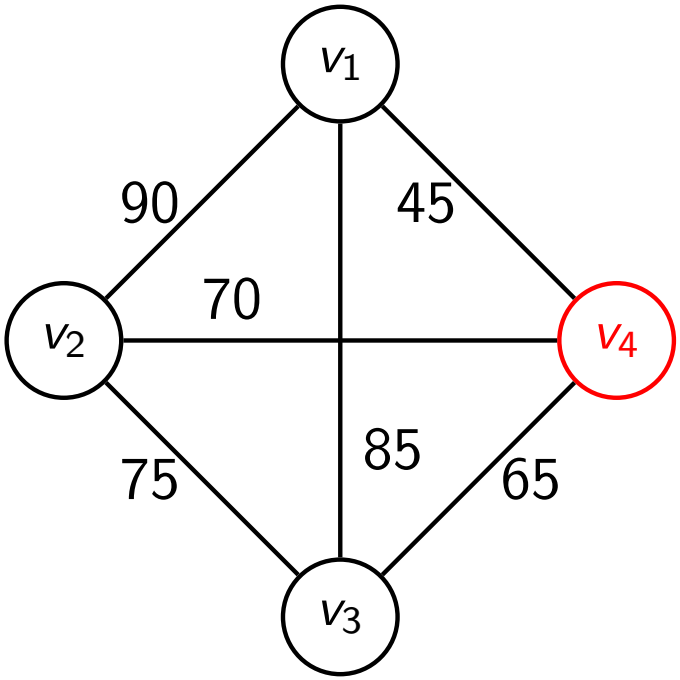
# Some Definitions

- Compute load: measure of overall load on the node
  - Static attributes: clock speed, core count, total memory.
  - Dynamic attributes: CPU load, CPU utilization, available memory
  - Compute load, $\mathrm{CL}_v = \sum_{a \in \text{attributes}} w_a * val_{va}$
- Network load: measure of load on the p2p network link
  - Latency
  - Bandwidth
  - Network load, $\mathrm{NL}_{(u,v)} = w_{lt}\mathrm{LT}_{(u,v)} + w_{bw}\mathrm{BW}_{(u,v)}$
- Avaliable processors: measure of effective number of processors
  - $pc_v = coreCount_v - Load_v \ \% \ coreCount_v$

# Some Definitions

- Compute load: measure of overall load on the node
  - Static attributes: clock speed, core count, total memory.
  - Dynamic attributes: CPU load, CPU utilization, available memory
  - Compute load, $\mathrm{CL}_v = \sum_{a \in \text{attributes}} w_a * val_{va}$
- Network load: measure of load on the p2p network link
  - Latency
  - Bandwidth
  - Network load, $\mathrm{NL}_{(u,v)} = w_{lt}\mathrm{LT}_{(u,v)} + w_{bw}\mathrm{BW}_{(u,v)}$
- Avaliable processors: measure of effective number of processors
  - $pc_v = coreCount_v - Load_v \,\%\, coreCount_v$
- Weights can be tuned according to program needs/type.

# Allocation Algorithm

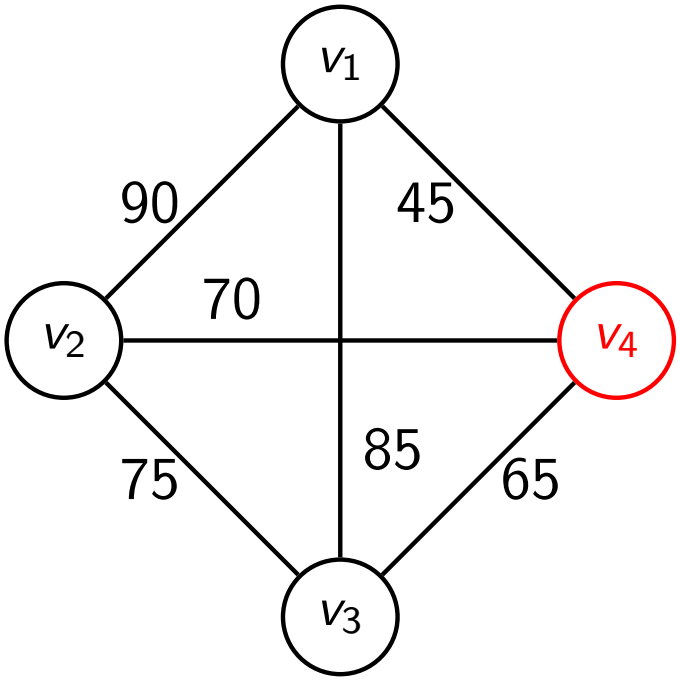



- Available core count of $v_4 = 4$
- Required process count = 16
- Network weight = 0.6
- Compute weight = 0.4

- Picked $v_4$
- Allocated process count = 4

| Node | Compute load | Network load | #Available cores |
|---|---|---|---|
| $v_1$ | 52 | 45 | 6 |
| $v_2$ | 47 | 70 | 8 |
| $v_3$ | 74 | 65 | 5 |

# Allocation Algorithm

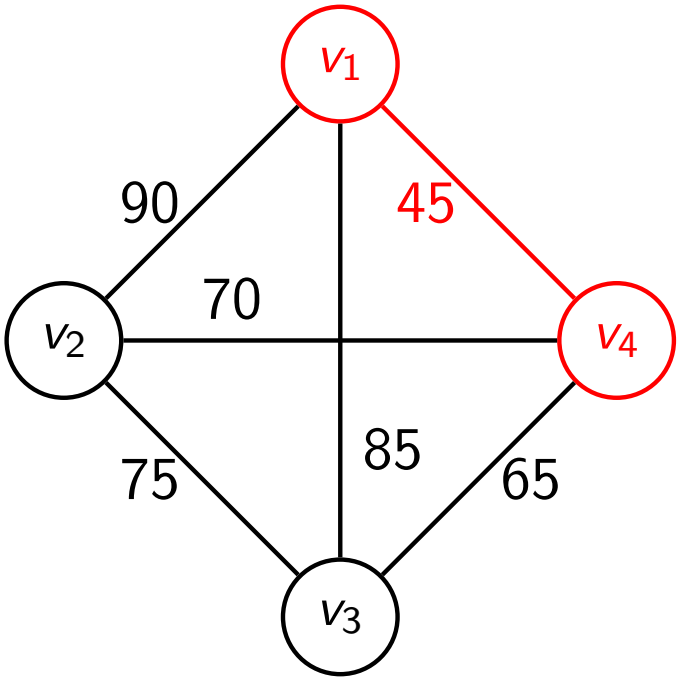



- Available core count of $v_4 = 4$
- Required process count = 16
- Network weight = 0.6
- Compute weight = 0.4

- Picked $v_4$
- Allocated process count = 4

| Node | Compute load | Network load | Addition load | #Available cores |
|---|---|---|---|---|
| $v_1$ | 52 | 45 | 47.8 | 6 |
| $v_2$ | 47 | 70 | 60.8 | 8 |
| $v_3$ | 74 | 65 | 68.6 | 5 |

# Allocation Algorithm

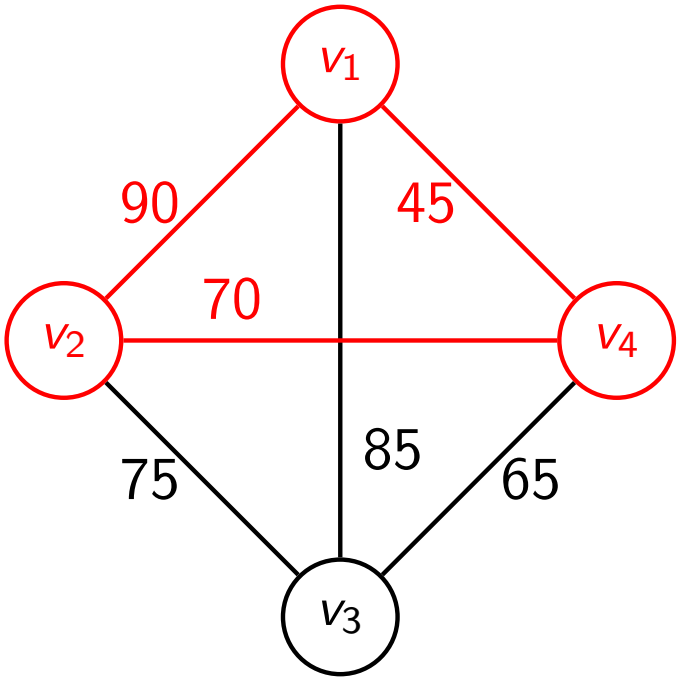



- Available core count of $v_4 = 4$
- Required process count = 16
- Network weight = 0.6
- Compute weight = 0.4

- Picked $v_1$
- Allocated process count = 10

| Node | Compute load | Network load | Addition load | #Available cores |
|---|---|---|---|---|
| $v_1$ | 52 | 45 | 47.8 | 6 |
| $v_2$ | 47 | 70 | 60.8 | 8 |
| $v_3$ | 74 | 65 | 68.6 | 5 |

# Allocation Algorithm




- Available core count of $v_4 = 4$
- Required process count = 16
- Network weight = 0.6
- Compute weight = 0.4

- Picked $v_2$
- Allocated process count = 18

| Node | Compute load | Network load | Addition load | #Available cores |
|---|---|---|---|---|
| $v_1$ | 52 | 45 | 47.8 | 6 |
| $v_2$ | 47 | 70 | 60.8 | 8 |
| $v_3$ | 74 | 65 | 68.6 | 5 |

# Allocation Algorithm

- Find candidate sub-graph corresponding to each node.

# Allocation Algorithm

- Find candidate sub-graph corresponding to each node.
- For each sub-graph $\mathcal{G}_v = (\mathcal{V}_v, \mathcal{E}_v)$ define:
  - Compute Load, $C_{G_v} = \sum_{u \in \mathcal{V}_v} CL_u$
  - Network Load, $N_{G_v} = \sum_{(x,y) \in \mathcal{E}_v} NL_{(x,y)}$

# Allocation Algorithm

- Find candidate sub-graph corresponding to each node.
- For each sub-graph $\mathcal{G}_v = (\mathcal{V}_v, \mathcal{E}_v)$ define:
  - Compute Load, $C_{G_v} = \sum_{u \in \mathcal{V}_v} CL_u$
  - Network Load, $N_{G_v} = \sum_{(x,y) \in \mathcal{E}_v} NL_{(x,y)}$
  - Total Load, $T_{G_v} = \alpha \times C_{G_v \text{ Normalized}} + \beta \times N_{G_v \text{ Normalized}}$

# Allocation Algorithm

- Find candidate sub-graph corresponding to each node.
- For each sub-graph $\mathcal{G}_v = (\mathcal{V}_v, \mathcal{E}_v)$ define:
  - Compute Load, $C_{G_v} = \sum_{u \in \mathcal{V}_v} CL_u$
  - Network Load, $N_{G_v} = \sum_{(x,y) \in \mathcal{E}_v} NL_{(x,y)}$
  - Total Load, $T_{G_v} = \alpha \times C_{G_v \text{ Normalized}} + \beta \times N_{G_v \text{ Normalized}}$
- Allocate the best one on the basis of total load

# Candidate Selection Algorithm

Candidate Selection Algorithm

- Compute addition cost for all the nodes w.r.t start node

$$\text{Addition Cost : } A_v(u) = \alpha \times CL(u) + \beta \times NL(v, u)$$

# Candidate Selection Algorithm

Candidate Selection Algorithm

- Compute addition cost for all the nodes w.r.t start node

$$\text{Addition Cost : } A_v(u) = \alpha \times CL(u) + \beta \times NL(v, u)$$

- Keep adding nodes to sub-graph until allocated processes is more than required number of processes

# Candidate Selection Algorithm

Input : Node $v$, Graph $\mathcal{G}$, $\mathcal{PC}$ List of effective processor count, $n$ Requested number of processes

Output: $\mathcal{G}_v$ sub-graph with $v$ included

# Candidate Selection Algorithm

Input : Node $v$, Graph $\mathcal{G}$, $\mathcal{PC}$ List of effective processor count, $n$ Requested number of processes

Output: $\mathcal{G}_v$ sub-graph with $v$ included

$\mathcal{V}_v \longleftarrow \phi$;

allocated process $\longleftarrow$ 0;

k $\longleftarrow$ Total number of nodes in $\mathcal{G}$;

# Candidate Selection Algorithm

Input : Node $v$, Graph $\mathcal{G}$, $\mathcal{PC}$ List of effective processor count, $n$ Requested number of processes

Output: $\mathcal{G}_v$ sub-graph with $v$ included

$\mathcal{V}_v \longleftarrow \phi$;

allocated process $\longleftarrow$ 0;

k $\longleftarrow$ Total number of nodes in $\mathcal{G}$;

$A_v(v) \longleftarrow 0$;

Calculate $A_v(u)$ for each node other than $v$;

Let $u_1$, $u_2$, $u_3$, ... , $u_k$ be the vertices sorted in increasing order of addition load $A_v(u)$;

# Candidate Selection Algorithm

Input : Node $v$, Graph $\mathcal{G}$, $\mathcal{PC}$ List of effective processor count, $n$ Requested number of processes
Output: $\mathcal{G}_v$ sub-graph with $v$ included

$\mathcal{V}_v \longleftarrow \phi$;
allocated process $\longleftarrow$ 0;
k $\longleftarrow$ Total number of nodes in $\mathcal{G}$;
$A_v(v) \longleftarrow 0$;
Calculate $A_v(u)$ for each node other than $v$;
Let $u_1$, $u_2$, $u_3$, ... , $u_k$ be the vertices sorted in increasing order of addition load $A_v(u)$;
i $\longleftarrow$ 1;
while *allocated processes* $< n$ do
    $\mathcal{V}_v \longleftarrow \mathcal{V}_v \cup \{u_i\}$;
    allocated process $\longleftarrow$ allocated process + $pc_{u_i}$;
    i $\longleftarrow$ i+1;
end

Algorithm 4: Candidate Selection Algorithm

# Resource Monitoring

- Developed a distributed monitoring system for flexibility

# Resource Monitoring

- Developed a distributed monitoring system for flexibility
- Light-weight deamons for periodically updating:

# Resource Monitoring

- Developed a distributed monitoring system for flexibility
- Light-weight deamons for periodically updating:
  - Livehosts: nodes which are up and running

# Resource Monitoring

- Developed a distributed monitoring system for flexibility
- Light-weight deamons for periodically updating:
  - Livehosts: nodes which are up and running
  - Node statistics: available memory, CPU load and CPU utilization, etc.

# Resource Monitoring

- Developed a distributed monitoring system for flexibility
- Light-weight deamons for periodically updating:
  - Livehosts: nodes which are up and running
  - Node statistics: available memory, CPU load and CPU utilization, etc.
  - Network statistics: available bandwidth and latency

# Resource Monitoring

- Developed a distributed monitoring system for flexibility
- Light-weight deamons for periodically updating:
  - Livehosts: nodes which are up and running
  - Node statistics: available memory, CPU load and CPU utilization, etc.
  - Network statistics: available bandwidth and latency
- Tracked average of last 1, 5, 15 minutes of data

# Resource Monitoring

- Developed a distributed monitoring system for flexibility
- Light-weight deamons for periodically updating:
  - Livehosts: nodes which are up and running
  - Node statistics: available memory, CPU load and CPU utilization, etc.
  - Network statistics: available bandwidth and latency
- Tracked average of last 1, 5, 15 minutes of data
- Stored data on Network File System

# Experimental Setup and Benchmark

- Experimental setup:
  - 40 12-core Intel Core nodes (4.6 GHz) and 20 8-core Intel Core nodes (2.8 GHz)
  - Cluster has a tree-like hierarchical topology with 4 switches

# Experimental Setup and Benchmark

- Experimental setup:
  - 40 12-core Intel Core nodes (4.6 GHz) and 20 8-core Intel Core nodes (2.8 GHz)
  - Cluster has a tree-like hierarchical topology with 4 switches
- Manveto benchmark (miniMD) : a simple, parallel molecular dynamics mini-application

# Experimental Setup and Benchmark

- Experimental setup:
  - 40 12-core Intel Core nodes (4.6 GHz) and 20 8-core Intel Core nodes (2.8 GHz)
  - Cluster has a tree-like hierarchical topology with 4 switches
- Manveto benchmark (miniMD) : a simple, parallel molecular dynamics mini-application
- Comparison with:
  - Random allocation
  - Sequential allocation
  - Load-aware allocation

# Weights for miniMD

| Attribute | Weight |
|---|---|
| CPU Load | 0.3 |
| CPU Utilization | 0.2 |
| Node Bandwidth | 0.2 |
| Used memory | 0.1 |
| Logical core count | 0.1 |
| Clock Speed | 0.05 |
| Total Memory | 0.05 |

Table: Relative weights for compute load

# Weights for miniMD

| Attribute | Weight |
|---|---|
| CPU Load | 0.3 |
| CPU Utilization | 0.2 |
| Node Bandwidth | 0.2 |
| Used memory | 0.1 |
| Logical core count | 0.1 |
| Clock Speed | 0.05 |
| Total Memory | 0.05 |

Table: Relative weights for compute load

- Relative weights for latency and bandwidth were set to 0.25 and 0.75 respectively.
- Relative weights for compute and network load were set to 0.3 and 0.7 respectively.

# Experiment: A specific run

Configuration: 32 processes, 4 processes per node

| Algorithm | Avg. Used Bandwidth | Avg. Latency | Avg. Compute Load |
|---|---|---|---|
| Random | 17.07 | 546.46 | 1.242 |
| Sequential | 10.72 | 304.25 | 1.262 |
| Load Aware | 18.64 | 354.51 | 0.453 |
| Our algorithm | 5.36 | 82.90 | 0.633 |

Table: Usage of allocated resource group before allocation

# Experiment: A specific run

Configuration: 32 processes, 4 processes per node

| Algorithm | Avg. Used Bandwidth | Avg. Latency | Avg. Compute Load |
|---|---|---|---|
| Random | 17.07 | 546.46 | 1.242 |
| Sequential | 10.72 | 304.25 | 1.262 |
| Load Aware | 18.64 | 354.51 | 0.453 |
| Our algorithm | 5.36 | 82.90 | 0.633 |

Table: Usage of allocated resource group before allocation

Total execution time:

- Random: 27.61s
- Sequential: 24.91s
- Load Aware: 12.31s
- Our Algorithm: 4.43s

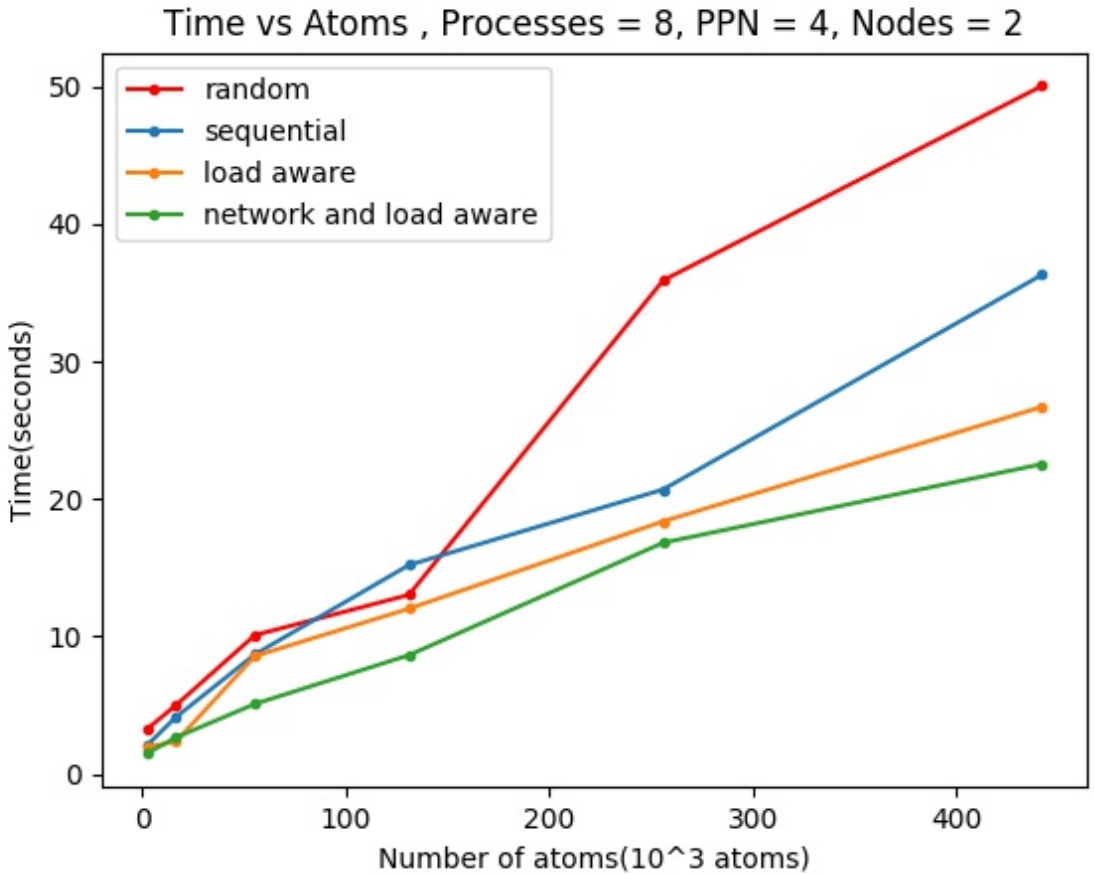
# Experiments: A specific run

| NODES | csews1 | csews4 | csews5 | csews6 | csews8 | csews9 | csews10 | csews12 | csews13 | csews15 | csews16 | csews19 | csews20 | csews23 | csews28 | csews32 | csews50 | csews51 | csews54 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| csews1 | 0 | 50 | 17 | 15 | 10 | 22 | 10 | 23 | 68 | 11 | 16 | 69 | 56 | 44 | 30 | 70 | 41 | 36 | 38 |
| csews4 | 50 | 0 | 29 | 24 | 14 | 12 | 25 | 21 | 32 | 10 | 9 | 80 | 71 | 45 | 29 | 73 | 61 | 67 | 52 |
| csews5 | 17 | 29 | 0 | 29 | 11 | 18 | 14 | 13 | 51 | 7 | 7 | 70 | 61 | 53 | 35 | 55 | 37 | 52 | 39 |
| csews6 | 15 | 24 | 29 | 0 | 15 | 15 | 9 | 9 | 55 | 21 | 31 | 75 | 65 | 48 | 38 | 59 | 41 | 55 | 51 |
| csews8 | 10 | 14 | 11 | 15 | 0 | 23 | 18 | 6 | 56 | 9 | 12 | 81 | 64 | 57 | 27 | 61 | 33 | 60 | 53 |
| csews9 | 22 | 12 | 18 | 15 | 23 | 0 | 11 | 5 | 50 | 9 | 6 | 75 | 37 | 40 | 55 | 57 | 22 | 55 | 56 |
| csews10 | 10 | 25 | 14 | 9 | 18 | 11 | 0 | 5 | 44 | 22 | 17 | 66 | 68 | 53 | 43 | 61 | 37 | 55 | 52 |
| csews12 | 23 | 21 | 13 | 9 | 6 | 5 | 5 | 0 | 32 | 6 | 9 | 71 | 64 | 52 | 43 | 71 | 41 | 60 | 53 |
| csews13 | 68 | 32 | 51 | 55 | 56 | 50 | 44 | 32 | 0 | 43 | 50 | 11 | 10 | 8 | 43 | 10 | 52 | 69 | 54 |
| csews15 | 11 | 10 | 7 | 21 | 9 | 9 | 22 | 6 | 43 | 0 | 10 | 65 | 66 | 54 | 39 | 45 | 50 | 65 | 54 |
| csews16 | 16 | 9 | 7 | 31 | 12 | 6 | 17 | 9 | 50 | 10 | 0 | 44 | 78 | 57 | 61 | 60 | 59 | 59 | 57 |
| csews19 | 69 | 80 | 70 | 75 | 81 | 75 | 66 | 71 | 11 | 65 | 44 | 0 | 13 | 29 | 11 | 10 | 48 | 72 | 38 |
| csews20 | 56 | 71 | 61 | 65 | 64 | 37 | 68 | 64 | 10 | 66 | 78 | 13 | 0 | 14 | 8 | 10 | 49 | 69 | 47 |
| csews23 | 44 | 45 | 53 | 48 | 57 | 40 | 53 | 52 | 8 | 54 | 57 | 29 | 14 | 0 | 10 | 24 | 43 | 77 | 40 |
| csews28 | 30 | 29 | 35 | 38 | 27 | 55 | 43 | 43 | 43 | 39 | 61 | 11 | 8 | 10 | 0 | 15 | 44 | 58 | 29 |
| csews32 | 70 | 73 | 55 | 59 | 61 | 57 | 61 | 71 | 10 | 45 | 60 | 10 | 10 | 24 | 15 | 0 | 47 | 72 | 38 |
| csews50 | 41 | 61 | 37 | 41 | 33 | 22 | 37 | 41 | 52 | 50 | 59 | 48 | 49 | 43 | 44 | 47 | 0 | 5 | 13 |
| csews51 | 36 | 67 | 52 | 55 | 60 | 55 | 55 | 60 | 69 | 65 | 59 | 72 | 69 | 77 | 58 | 72 | 5 | 0 | 9 |
| csews54 | 38 | 52 | 39 | 51 | 53 | 56 | 52 | 53 | 54 | 54 | 57 | 38 | 47 | 40 | 29 | 38 | 13 | 9 | 0 |

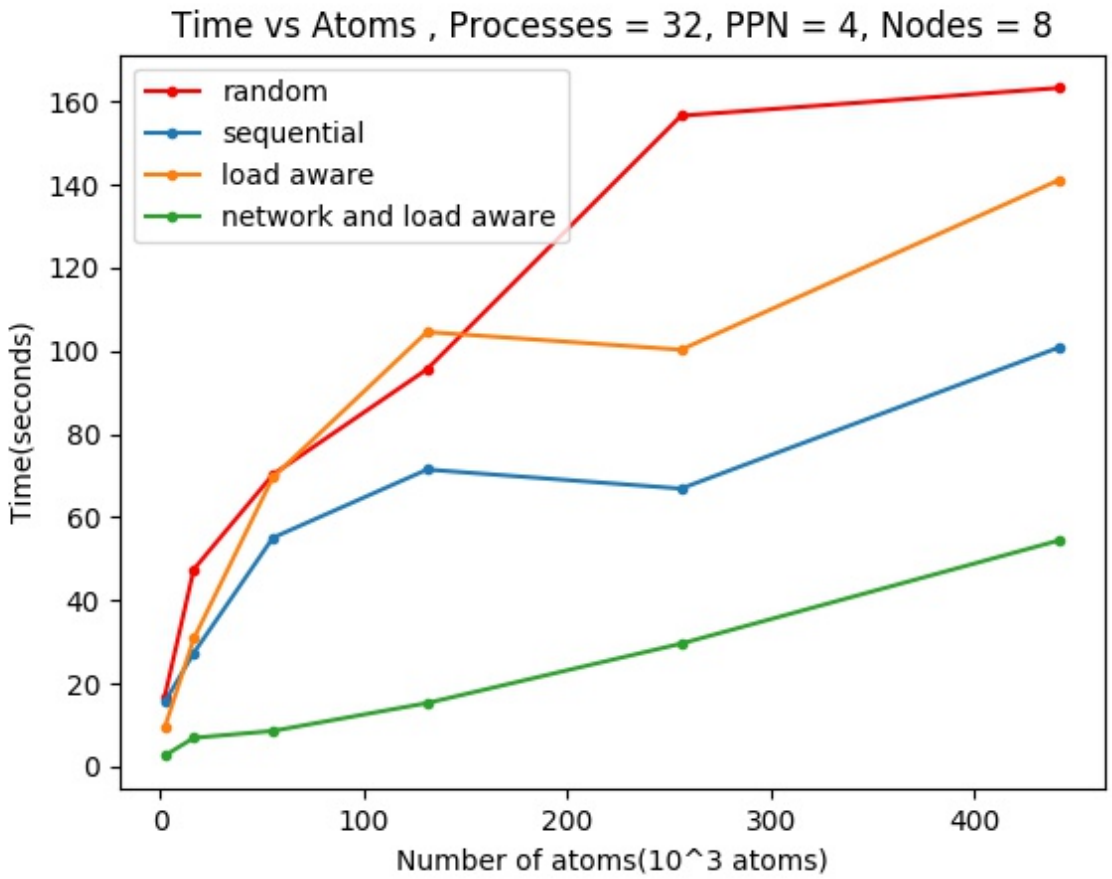
| | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Our | csews1 | csews4 | csews5 | csews6 | csews8 | csews9 | csews10 | csews12 | csews13 | csews15 | csews16 | csews19 | csews20 | csews23 | csews28 | csews32 | csews50 | csews51 | csews54 |
| Load | csews1 | csews4 | csews5 | csews6 | csews8 | csews9 | csews10 | csews12 | csews13 | csews15 | csews16 | csews19 | csews20 | csews23 | csews28 | csews32 | csews50 | csews51 | csews54 |
| Seq | csews1 | csews4 | csews5 | csews6 | csews8 | csews9 | csews10 | csews12 | csews13 | csews15 | csews16 | csews19 | csews20 | csews23 | csews28 | csews32 | csews50 | csews51 | csews54 |
| Random | csews1 | csews4 | csews5 | csews6 | csews8 | csews9 | csews10 | csews12 | csews13 | csews15 | csews16 | csews19 | csews20 | csews23 | csews28 | csews32 | csews50 | csews51 | csews54 |
| Load | 0.68 | 5.19 | 0.58 | 0.68 | 0.66 | 1.08 | 0.71 | 0.24 | 0.78 | 0.56 | 0.55 | 0.7 | 0.38 | 1.1 | 0.62 | 0.54 | 0.43 | 0.35 | 0.42 |

Figure: Peer-to-peer bandwidth and CPU Load

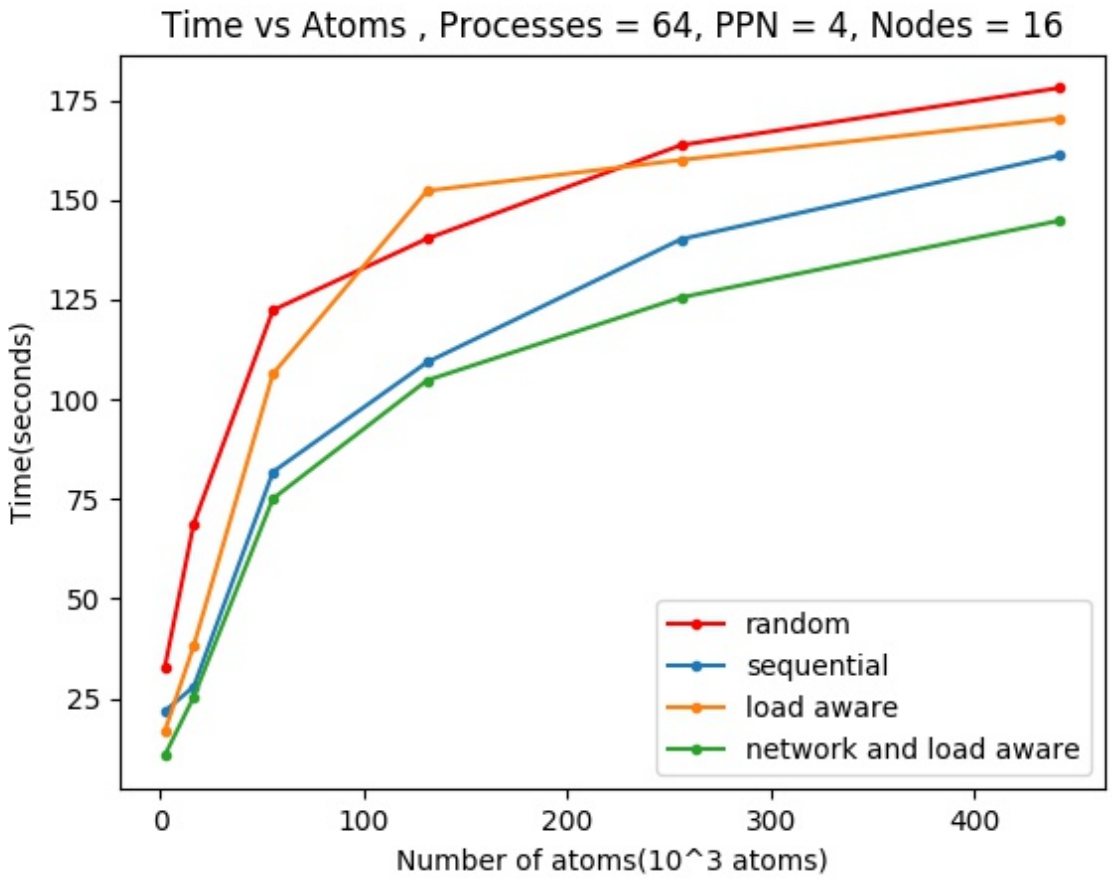
# Experiments



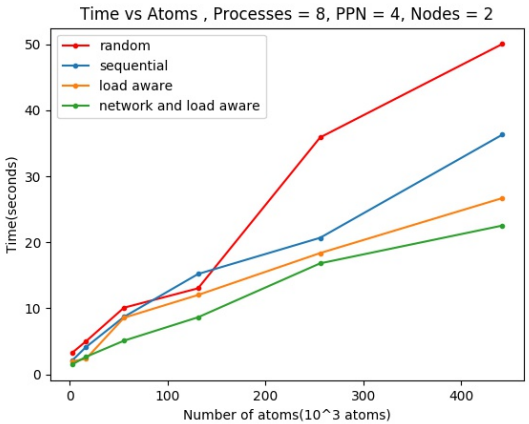
# Experiments

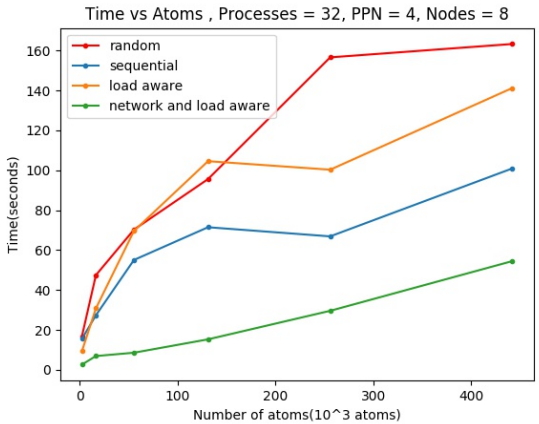


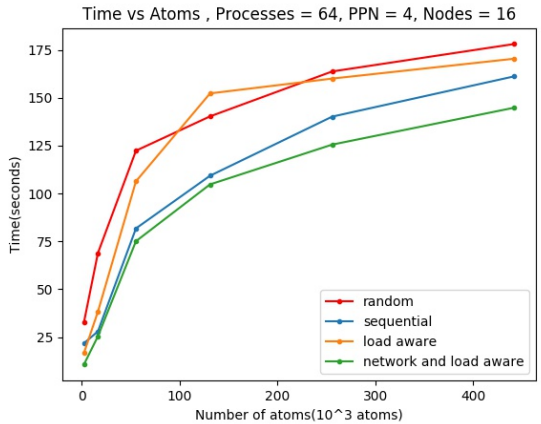
# Experiments



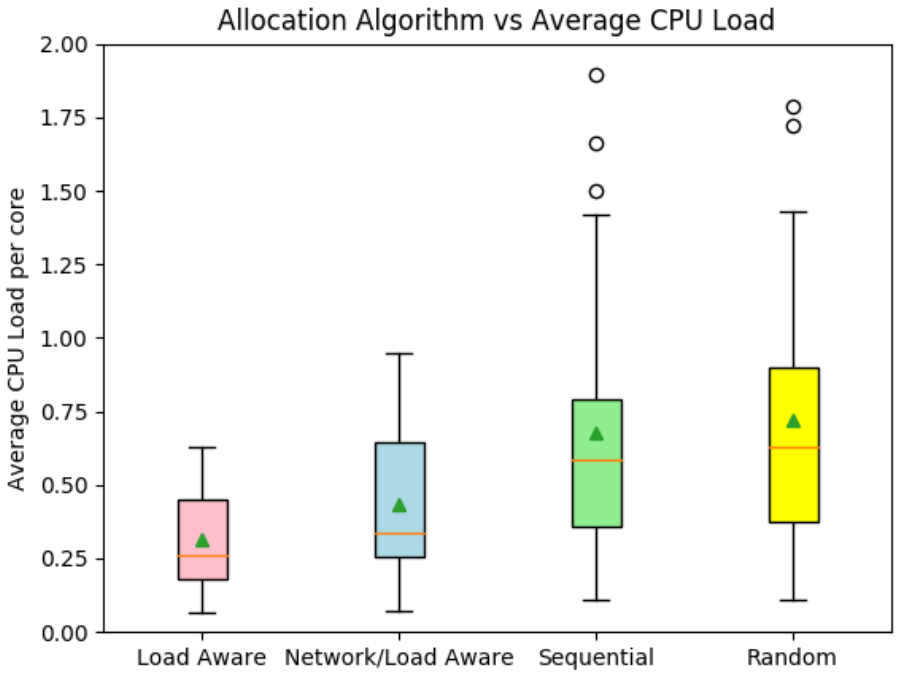
# Experiments



# Experiments




Average CPU load per logical core:

- Load aware = 0.31
- Network and Load aware = 0.43
- Sequential = 0.68
- Random = 0.78

# Performance Gain

| Algorithm | Avg. improvement | Max. improvement |
|---|---|---|
| Random | 49.9% | 87.8% |
| Sequential | 43.1% | 84.5% |
| Load Aware | 32.4% | 87.7% |

# Shortcomings and Future Work

- Benchmark testing on Mantevo benchmarks.

# Shortcomings and Future Work

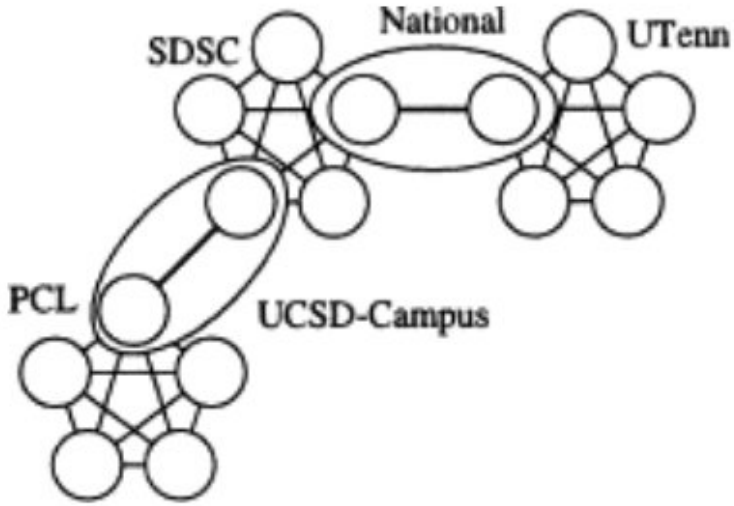
- Benchmark testing on Mantevo benchmarks.
- Need for formalization of weights estimation.

# Shortcomings and Future Work

- Benchmark testing on Mantevo benchmarks.
- Need for formalization of weights estimation.
- Time series estimation methods may be used for bandwidth forecast.

# Shortcomings and Future Work

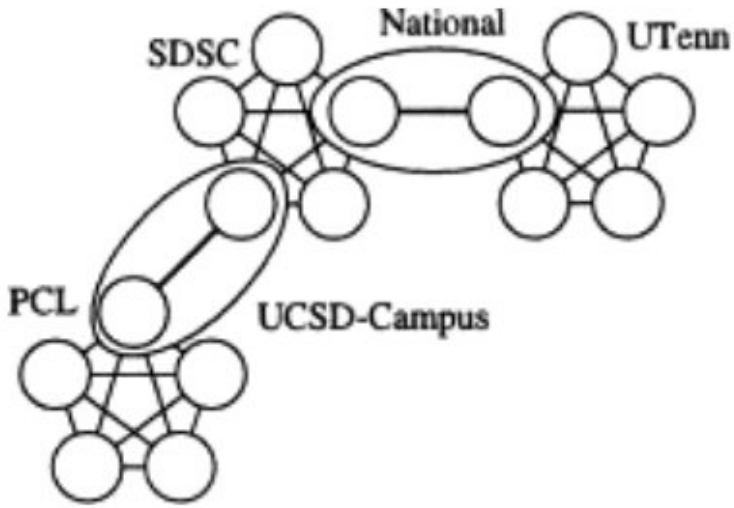
- Benchmark testing on Mantevo benchmarks.
- Need for formalization of weights estimation.
- Time series estimation methods may be used for bandwidth forecast.
- Extension to large scale systems, spanning over multiple clusters.




Source: *A Grid Resource Broker with Network Bandwidth-Aware Job Scheduling for Computational Grids*

# Shortcomings and Future Work

- Benchmark testing on Mantevo benchmarks.
- Need for formalization of weights estimation.
- Time series estimation methods may be used for bandwidth forecast.
- Extension to large scale systems, spanning over multiple clusters.

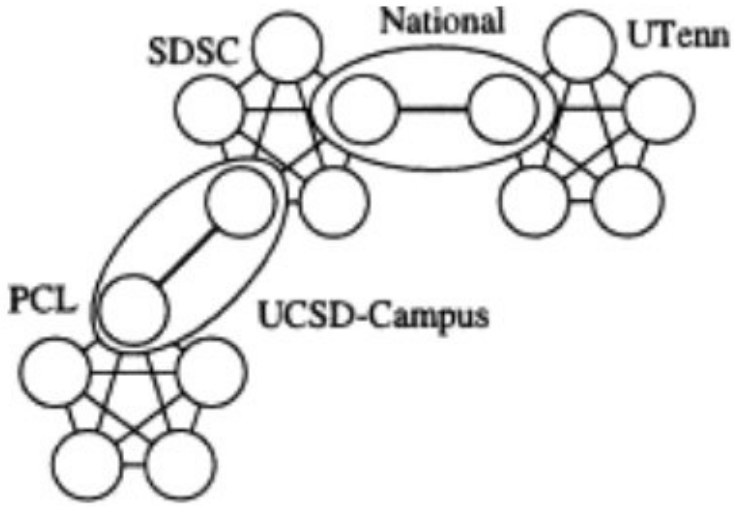



Source: *A Grid Resource Broker with Network Bandwidth-Aware Job Scheduling for Computational Grids*

- Resource allocation considering the communication pattern of the program.

# Shortcomings and Future Work

- Benchmark testing on Mantevo benchmarks.
- Need for formalization of weights estimation.
- Time series estimation methods may be used for bandwidth forecast.
- Extension to large scale systems, spanning over multiple clusters.




Source: *A Grid Resource Broker with Network Bandwidth-Aware Job Scheduling for Computational Grids*

- Resource allocation considering the communication pattern of the program.

*We would be presenting our work as a poster in 12th HiPC Student Research Symposium.*

# References

M. Kaur and S. S. Kadam, *"Discovery of resources using MADM approaches for parallel and distributed computing,"*Engineering Science and Technology, an International Journal, vol. 20, no. 3,pp. 1013 – 1024, 2017.

C.-T. Yang and et al., *"A Grid Resource Broker with Network Bandwidth-Aware Job Scheduling for Computational Grids,"* in Advances in Grid and Pervasive Computing, 2007

M. Alicherry and T. V. Lakshman, *"Network aware resource allocation in distributed clouds,"* in 2012 Proceedings IEEE INFOCOM, 2012, pp. 963–971.

A. B. Yoo, M. A. Jette, and M. Grondona, *"Slurm: Simple linux utility for resource management,"* in Job Scheduling Strategies for Parallel Processing, 2003, pp. 44–60.