A Graph Partitioning Game for Distributed Simulation of Networks

Aditya Kurve, Christopher Griffin, David Miller and George Kesidis

The Pennsylvania State University

Sept 9, 2011

Outline

1 Introduction

2 Background

3 Problem Setup

4 Iterative Refinement Steps

5 Software Simulator Platform

6 Future Work

7 Summary

Outline

1 Introduction

2 Background

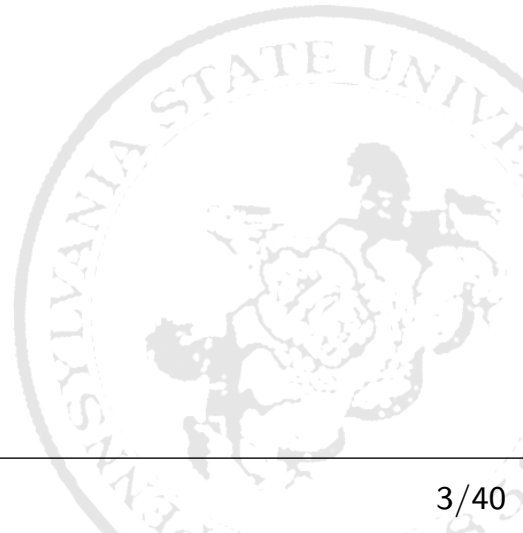
3 Problem Setup

4 Iterative Refinement Steps

5 Software Simulator Platform

6 Future Work

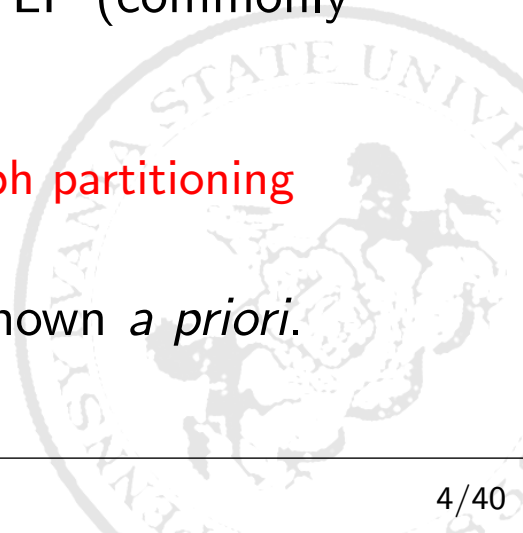
7 Summary



Definitions and Terminology

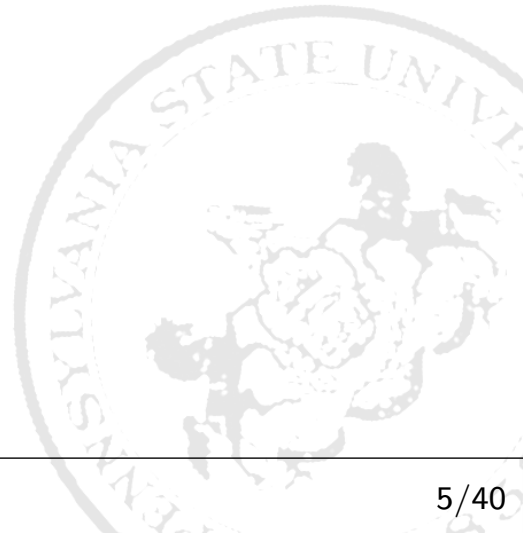
- Network Model: This is the simulation model of the network under study. The nodes represent routers, autonomous systems, buffers, clients, servers... etc.
- Machines: Refers to the hosts or processors or computers that are part of the simulator hardware.
- Logical Process
  - Logical Process (LP) : Fundamental unit of execution which runs in parallel and shares a machine with other LPs.
  - Every LP has a set of variable that define its state.
  - Controls these local variables and communicates with other LPs via messages.
  - Model each node in simulated network model as one LP (commonly used method).

- What makes this problem different from the well-known graph partitioning problem?
  1. Graph properties (Node and edge weights) are not known *a priori*.
  2. Graph properties change as simulation progresses.



Desired Features for the Partitioning Algorithm

- Involves all machines (exploit parallelism of machines).
- Uses distributed approach for partitioning (with minimal global information i.e., mainly local information exchanged among machines).
- Should converge to an equilibrium solution that is optimal in some sense if the simulation task is static.
- Iteratively improves partition with the dynamics of the graph of LPs (avoid complete refresh each time).



Outline

1 Introduction

2 Background

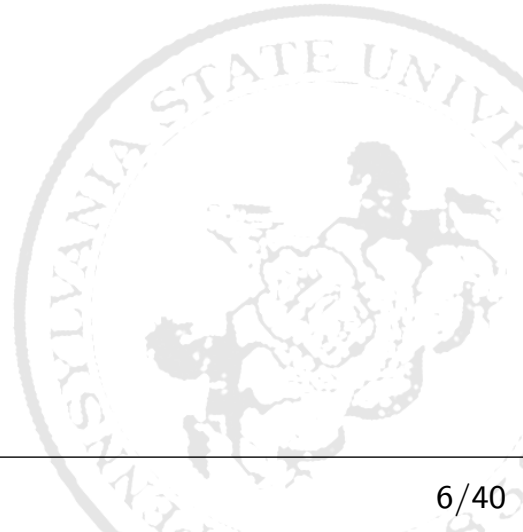
3 Problem Setup

4 Iterative Refinement Steps

5 Software Simulator Platform

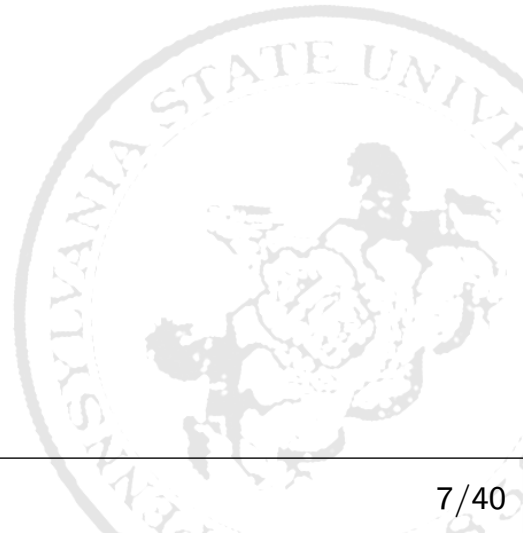
6 Future Work

7 Summary



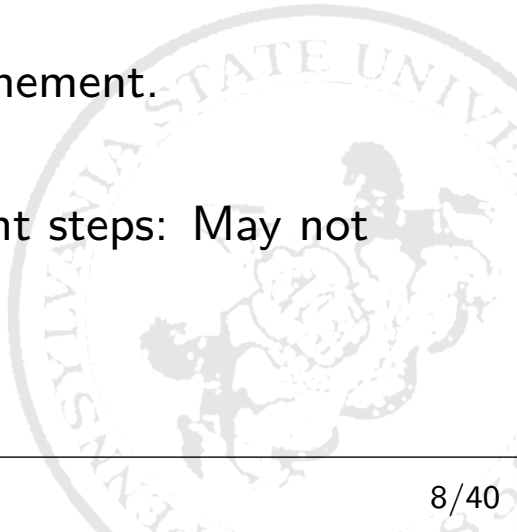
Distributed Network Simulation

- Represent each node in a simulated network by a logical process (LP) giving a graphical model of LPs.
- Two types of synchronization between LPs operating in parallel:
- **Conservative Synchronization:** LPs follow time causality strictly.
- Each link ensures that messages are sent in the order of their time stamps.
- **Optimistic Synchronization:** LP processes events ahead of time without any kind of assurance.
- In case it receives an event time stamped lower, it roll backs in time.
- **Balancing Load** among machines.



Centralized Graph Partitioning Problem

- Given a graph $G = (V, E)$ where V is the set of nodes and E is the set of edges, suppose the nodes and the edges are weighted and let w_i represent the weight of the i^th node and let C_{ij} represent the weight of edge (i, j).
- The K-way graph partitioning problem aims to find K subsets of V, $V_1, V_2, .., V_K$ such that $V_i \cap V_j = \emptyset \ \forall \ i, j$ and $\bigcup_{i=1}^{K} = V$, $\sum_{j \in V_i} w_j = \frac{\sum_k w_k}{K} \ \forall \ j$ and the sum of the weights of edges whose incident vertices belong to different subsets is minimized. NP-complete.
- Some heuristics:
  - Spectral bisection methods [Pothen *et al.* 1990]:
  - Geometric Methods for special graphs.
  - Multilevel Partitioning [Karypis 1996]:
    - Coarsen the graph $\rightarrow$ partition $\rightarrow$ uncoarsen with refinement.
    - Use Kernighan-Lin method for refinement.
- Mostly centralized and do not give incrementally improvement steps: May not be suitable for dynamic graphs.

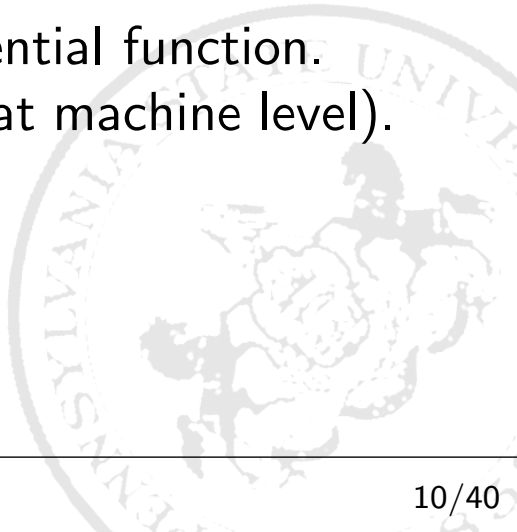


Our local strategies for iterative improvement

- Machines own nodes (LPs).
- Node by node exchange for improving upon existing partition.
- We'll describe two criteria for node transfer:
  1. Framework 1: Start with local cost function.
     - Design local cost for each node i as C_i.
     - Prove existence of Nash Equilibrium in pure strategies.
     - Prove convergence for synchronized iterative improvement by finding an energy function C_0 decreasing with the iterative improvements.
  2. Framework 2: Start with global cost function.
     - Define a global cost function $\tilde{C}_0$.
     - Derive a local cost $\tilde{C}_i$ at each node i from the global cost function $\tilde{C}_0$.
     - All the local minima of $\tilde{C}_0$ are the Nash equilibrium points and $\tilde{C}_0$ is the energy function of the game.
- Note that $\tilde{C}_i \neq C_i$.
- Both have comparable communication overhead needed for exchange of local information which is scalable since it is at machine-level and not at node-level.
- We compare simulation time for the two approaches.

Game Theoretic Approach

- **Selfish Load Balancing:**
  - Existing "Algorithmic Game Theory" approaches form a game where tasks (LPs) choose their machines.
  - Objective: To minimize the maximum load across machines.
  - Typically, communication costs (correlated with the threat of rollback) are not considered.
- **Our Approach:**
  - Factor inter-machine communication cost, so more general graph partitioning instead of load balancing.
  - We use different objective (cost) function for each node.
  - Local cost that decreases in a Lyapunov/energy/potential function.
  - Local cost can be calculated using local parameters (at machine level).



Outline

1 Introduction

2 Background

3 Problem Setup

4 Iterative Refinement Steps

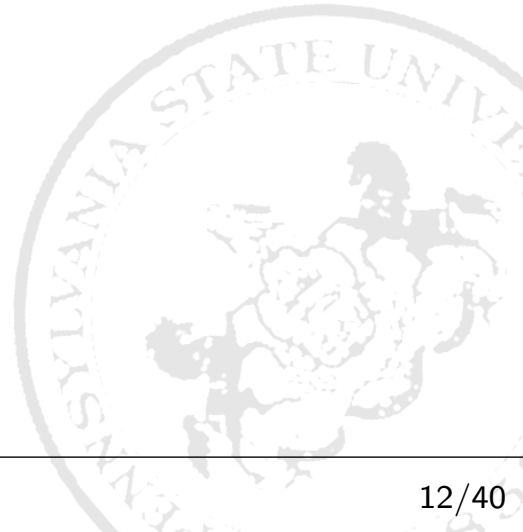
5 Software Simulator Platform

6 Future Work

7 Summary

Problem Setup

- A connected graph $G = (V, E)$ and K machines.
- b_i represent the computational load of i^{th} node.
- c_{ij} denote the cost of communicating over the edge (i, j).
- G is to be partitioned between K machines.
- w_k be the normalized capacity/speed of the k^{th} machines so that $\sum_k w_k = 1$.
- Formulate game with machines as players [CAMAD11].
- Game with nodes as players: Each node chooses a machines based on a cost function.



What cost function should a node use?

- N nodes and K machines. Let $r_i \in \{1, 2, ...K\}$ be the machines chosen by the i^{th} node.

$$C_i(r_i, \mathbf{r}_{-i}) = \frac{b_i}{w_{r_i}} \sum_{\substack{j:r_j=r_i \\ j\neq i}} b_j + \frac{\mu}{2} \sum_{j:r_j\neq r_i} c_{ij},$$

What cost function should a node use?

- N nodes and K machines. Let $r_i \in \{1, 2, ...K\}$ be the machines chosen by the i^{th} node.

$$C_i(r_i, \mathbf{r}_{-i}) = \frac{b_i}{w_{r_i}} \sum_{\substack{j:r_j=r_i \\ j\neq i}} b_j + \frac{\mu}{2} \sum_{j:r_j\neq r_i} c_{ij},$$

Payoff depends on decision by other nodes

What cost function should a node use?

- N nodes and K machines. Let $r_i \in \{1, 2, ...K\}$ be the machines chosen by the i^{th} node.

$$C_i(r_i, \mathbf{r}_{-i}) = \frac{b_i}{w_{r_i}} \sum_{\substack{j:r_j=r_i \\ j \neq i}} b_j + \frac{\mu}{2} \sum_{j:r_j \neq r_i} c_{ij},$$

Payoff depends on decision by other nodes

Load brought by node i to machine

What cost function should a node use?

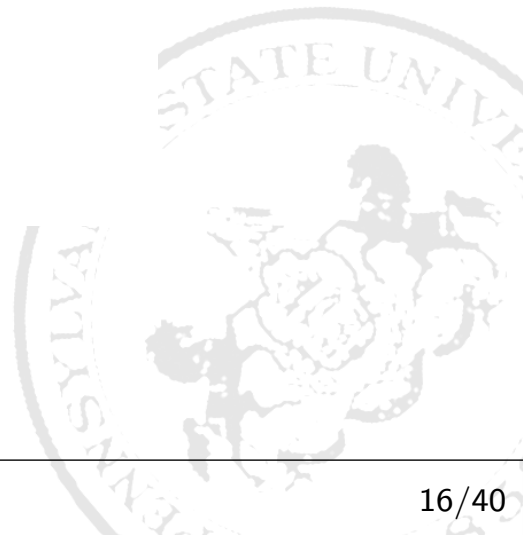
- N nodes and K machines. Let $r_i \in \{1, 2, ...K\}$ be the machines chosen by the i^{th} node.

$$C_i(r_i, \mathbf{r}_{-i}) = \frac{b_i}{w_{r_i}} \sum_{\substack{j:r_j=r_i \\ j \neq i}} b_j + \frac{\mu}{2} \sum_{j:r_j \neq r_i} c_{ij},$$

Payoff depends on decision by other nodes

Load brought by node i to machine

Load already present on the machine



What cost function should a node use?

- N nodes and K machines. Let $r_i \in \{1, 2, ...K\}$ be the machines chosen by the i^{th} node.

$$C_i(r_i, \mathbf{r}_{-i}) = \frac{b_i}{w_{r_i}} \sum_{\substack{j:r_j=r_i \\ j \neq i}} b_j + \frac{\mu}{2} \sum_{j:r_j \neq r_i} c_{ij},$$

Payoff depends on decision by other nodes

Load brought by node i to machine

Load already present on the machine

Communication cost with neighbors on other machines

What cost function should a node use?

- N nodes and K machines. Let $r_i \in \{1, 2, ...K\}$ be the machines chosen by the i^{th} node.

$$C_i(r_i, \mathbf{r}_{-i}) = \frac{b_i}{w_{r_i}} \sum_{\substack{j:r_j=r_i \\ j\neq i}} b_j + \frac{\mu}{2} \sum_{j:r_j\neq r_i} c_{ij},$$

Payoff depends on decision by other nodes

Load brought by node i to machine

Load already present on the machine

Communication cost with neighbors on other machines

- μ is the relative weight given to the inter-machine communication compared to computation. For remotely connected machines, it might be large.

Nash equilibrium for the game

- A strategy profile $\mathbf{r} = (r_1^*, r_2^*, ..., r_N^*)$ is a Nash equilibrium iff

$$C_i(r_i^*, \mathbf{r}_{-i}^*) \leq C_i(r_i, \mathbf{r}_{-i}^*) \quad \forall r_i \in \{1, 2, ..., K\} \; \forall i.$$

- **Theorem**

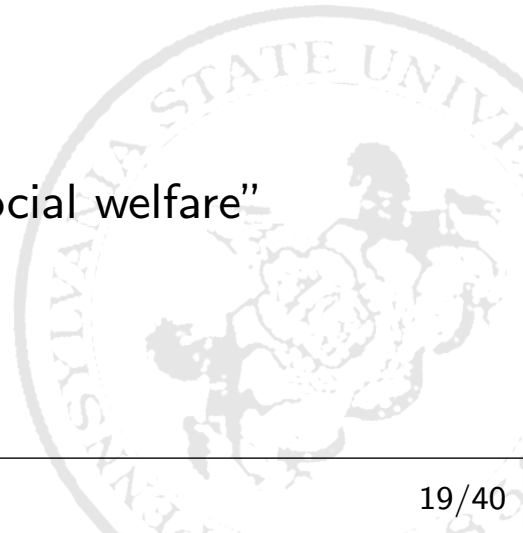
For the game described above, a Nash equilibrium exists in pure strategies.

- The solution to the optimization problem

$$\min_{\mathbf{r}} \; C_0(\mathbf{r}) := \sum_i \left(\frac{b_i}{w_{r_i}} \sum_{\substack{j: r_j = r_i \\ j \neq i}} b_j + \frac{\mu}{2} \sum_{j: r_j \neq r_i} c_{ij} \right).$$

  is also a Nash equilibrium for the game.

- C_0 is the sum of the costs of all the nodes interpreted as "social welfare" function.



Proof Sketch

- We prove that when the node changes machines to improve its cost, the value of $C_0(\mathbf{r})$ decreases.
- Hence our assumption that it is solution to optimization problem is wrong.

$$\begin{aligned} C_0(\mathbf{r}) = & C_0(\mathbf{r}|i = l) + C_0(\mathbf{r}|i \neq l, r_i = k_1) + C_0(\mathbf{r}|i \neq l, r_i = k_2) \\ & + C_0(\mathbf{r}|i \neq l, r_i \neq k_1, r_i \neq k_2) \end{aligned}$$

- Suppose there exists a better assignment $r_l^* \neq \hat{r}_l$ for a node l, *i.e.*, the l^{th} node can decrease its cost by moving from $\hat{r}_l$ machines to r_l^* machines. Then the new assignment vector is $\mathbf{r}^* = (r_1^*, r_2^*, ..., r_N^*)$ and $C_l(\mathbf{r}^*) - C_l(\hat{\mathbf{r}}) < 0$.

$$\begin{aligned} C_0(\mathbf{r}^*) - C_0(\hat{\mathbf{r}}) = & C_0(\mathbf{r}^*|i = l) - C_0(\hat{\mathbf{r}}|i = l) + \\ & + C_0(\mathbf{r}^*|i \neq l, r_i^* \neq \hat{r}_l, r_i^* \neq r_l^*) - C_0(\hat{\mathbf{r}}|i \neq l, \hat{r}_i \neq \hat{r}_l, \hat{r}_i \neq r_l^*) \\ & + C_0(\mathbf{r}^*|i \neq l, r_i^* = \hat{r}_l) - \Psi(\hat{\mathbf{r}}|i \neq l, \hat{r}_i = \hat{r}_l) \\ & + C_0(\mathbf{r}^*|i \neq l, r_i^* = r_l^*) - \Psi(\hat{\mathbf{r}}|i \neq l, \hat{r}_i = r_l^*) \end{aligned}$$

Outline

1 Introduction

2 Background

3 Problem Setup

4 Iterative Refinement Steps

5 Software Simulator Platform

6 Future Work

7 Summary

Initial Partitioning

- Initial node to host assignment with connected partitions.
  - Each machine identifies a unique node called focal node.
  - Random hop-by-hop expansion from the focal node.
  - At this time nodes and edge weights are not known.

- Finding focal nodes
  - Idea: Focal nodes should be as far apart as possible (geodesic distance).
  - For a random graph, find focal nodes with mean minimum distance between any two pairs of focal nodes being $N_{\frac{|V|}{K}}$ the mean number of hops that cover $\frac{|V|}{K}$ nodes, then the partitions will be more or less equal.
  - So if $F = \{f_1, f_2, ..., f_K\}$ is the set of focal nodes, then we ideally want

$$F = \underset{H \subseteq V \ s.t. \ |H|=K}{\arg\max} \ \min_{h,l \in H : l \neq h} d_G(h, l),$$

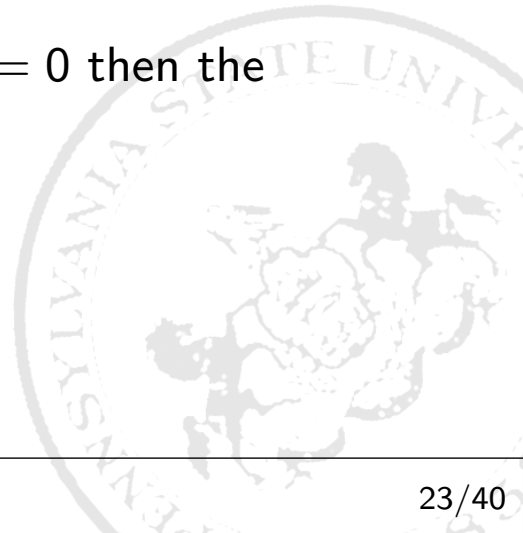
    where d_G is the geodesic distance between the two nodes.

Local and Synchronous Iterative Partition Refinements

- Considering dynamic graph of LPs, iterative improvements to correct the load imbalances at regular time intervals instead of a complete refresh of the partition.
- Each machine takes turns to decide the "most dissatisfied" node in its partition and transfers its ownership to a new machine.
- Dissatisfaction of the i^{th} node defined as

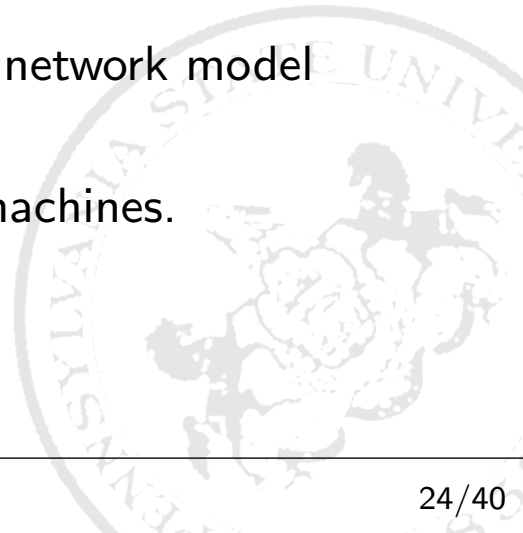
$$\Im(i) = C_i(r_i) - \min_k \; C_i(k)$$

- The most dissatisfied node with a machine is the one with maximum value of $\Im$.
- $\Im = 0$ for the most dissatisfied node, then the machine forsakes its turn.
- When all the machines have most dissatisfied nodes with $\Im = 0$ then the algorithm has converged.



Inter-machine communication overhead for calculating $\Im(i)$

- To calculate $\min_k \ C_i(k)$ we need to know the sum of weights of nodes at other machines $\sum_{\substack{j:r_j=k \\ j\neq i}} b_j = \sum_{j:r_j=k} b_j$.
- Do not need to know the allocation for each node, i.e., no communication overhead at node-level.
- Note that the amount of packets exchanged to synchronize this parameter defining aggregate "state" scales linearly with the number of machines and is constant with respect to the number of nodes.
- Synchronization cost is independent of the size of simulated network model graph.
- Every iterative step followed with synchronization between machines.



Convergence of Algorithm

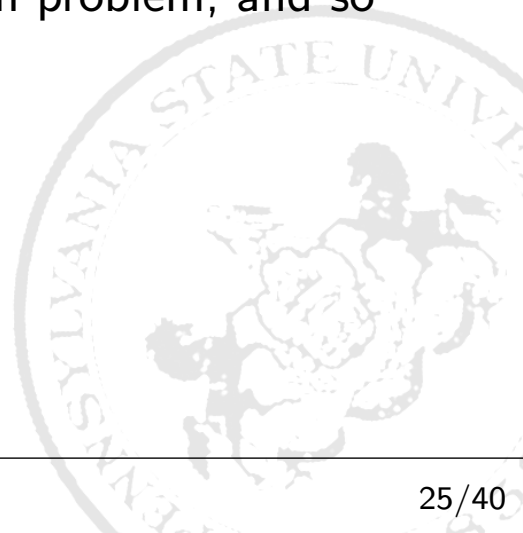
- Suppose the I^{th} node is transferred from its current allocated machines r_I to a new machines r_I^*. Call the new assignment vector $\mathbf{r}^*$. Then obviously,

$$C_I(\mathbf{r}^*) - C_I(\mathbf{r}) < 0.$$

- We can show from the theorem of existence of Nash equilibrium,

$$C_0(\mathbf{r}^*) - C_0(\mathbf{r}) = 2\big(C_I(\mathbf{r}^*) - C_I(\mathbf{r})\big) < 0$$

- Hence for every iterative step the potential function $C_0(\mathbf{r})$ decreases.
- Bounded nature of solution to the combinatorial optimization problem, and so achievable lower bound exists.



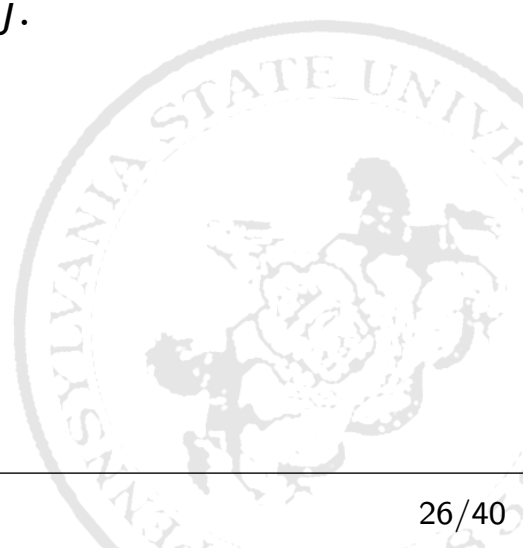
Defining centralized cost: An alternative cost framework

- We now give an alternative cost framework.
- Centralized Partitioning Cost:

$$\min \quad \tilde{C}_0 = \sum_{k=0}^{K} \left(\frac{\sum_{j \in V} x_{kj} b_j}{w_k} - \sum_j b_j \right)^2 + \frac{\mu}{2} \sum_{i,j} c_{ij} x_{ki} (1 - x_{kj})$$

$$\textit{subject to} \ \sum_k x_{kj} = 1 \ \forall \ j \ \textit{and} \ x_{kj} \in \{0, 1\} \ \forall \ k, j.$$

- May not be convex depending on the graph topology.



Improving cost: one node at a time

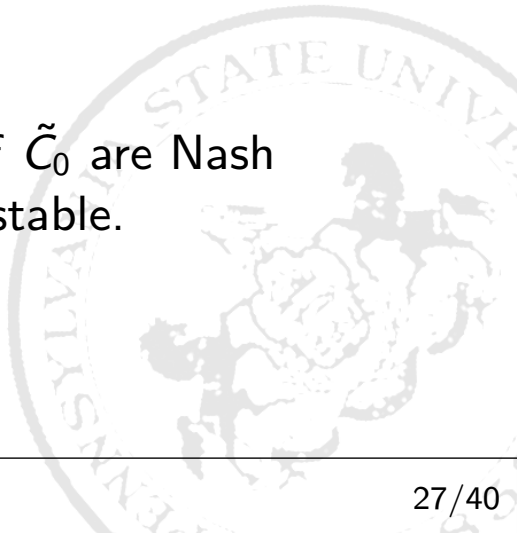
- Suppose a node i is moved from machine m to machine n.
- Let $\tilde{C}_0^{new}$ be the new value of cost function.
- We can show that

$$\tilde{C}_0 - \tilde{C}_0^{new} = \left(\frac{b_i^2}{w_m^2} + \frac{2b_i}{w_m^2}\sum_{\substack{j:r_j=m \\ j\neq i}} b_j - \frac{2b_i}{w_m}\sum_j b_j\right) - \left(\frac{b_i^2}{w_n^2} + \frac{2b_i}{w_n^2}\sum_{\substack{j:r_j=n \\ j\neq i}} b_j - \frac{2b_i}{w_n}\sum_j b_j\right)$$

- So we can define a new cost function for each node as:

$$\tilde{C}_i(r_i, \mathbf{r}_{-i}) = \frac{b_i^2}{w_{r_i}^2} + \frac{2b_i}{w_{r_i}^2}\sum_{\substack{j:r_j=r_i \\ j\neq i}} b_j - \frac{2b_i}{w_{r_i}}\sum_j b_j$$

- Nash equilibrium exists and is stable: All the local minima of $\tilde{C}_0$ are Nash equilibria for the game with this node cost function and are stable.



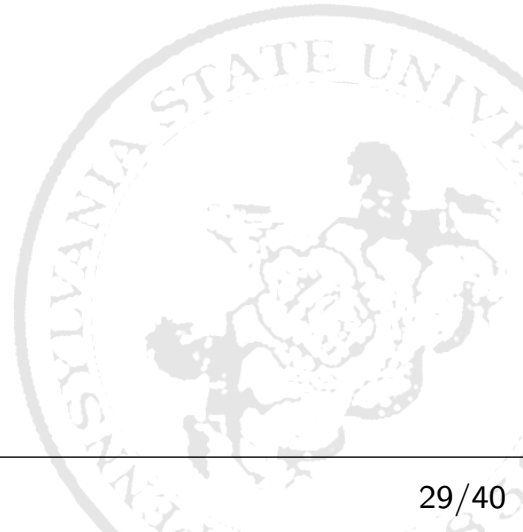
Comparing the iterative refinement at equilibria with the two cost frameworks

- We used NetLogo (multi-agent simulation software) to create random graph of 230 nodes representing graph of LPs to be divided into 5 partitions.
- We randomly generated node and edge weights each with a mean 5.
- The normalized machine speed (w_k) were 0.1, 0.2, 0.3, 0.3, 0.1 and $\mu = 8$.
- The degree of each node varied from 3 to 6 randomly.
- Convergence when no more improvement in C_0 (or $\tilde{C}_0$).

Random Graph Realization	Using $C_i(r_i, \underline{r}_{-i})$ for iterative refinement			Using $\tilde{C}_i(r_i, \underline{r}_{-i})$ for iterative refinement		
Case/ Trial Number	C_0	$\tilde{C}_0$	No. of Iterations to converge	C_0	$\tilde{C}_0$	No. of Iterations to converge
1	457134	4363	42	463130	6692	40
2	461704	4826	99	471539	9405	79
3	456260	2920	29	461614	5300	24
4	472157	2451	15	475814	3976	8
5	456336	3322	69	463559	6390	51

Observations: Comparison of two cost frameworks

- For all trials, $\tilde{C}_0$ does better under C_i refinement and so does C_0.
- Hence the two cost frameworks are inherently different.
- Also when iterating:
  - using C_i, $\tilde{C}_0$ may not monotonically decrease.
  - using $\tilde{C}_i$, C_0 may not monotonically decrease.
- Hence iterative improvement using C_i gives better steady state values not just for C_0 but also for $\tilde{C}_0$.



Outline

1 Introduction

2 Background

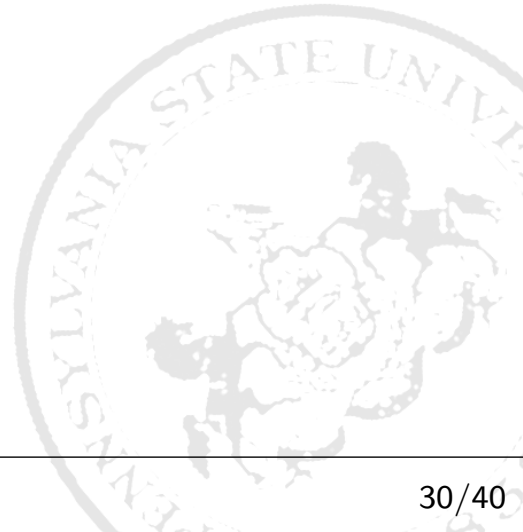
3 Problem Setup

4 Iterative Refinement Steps

5 Software Simulator Platform

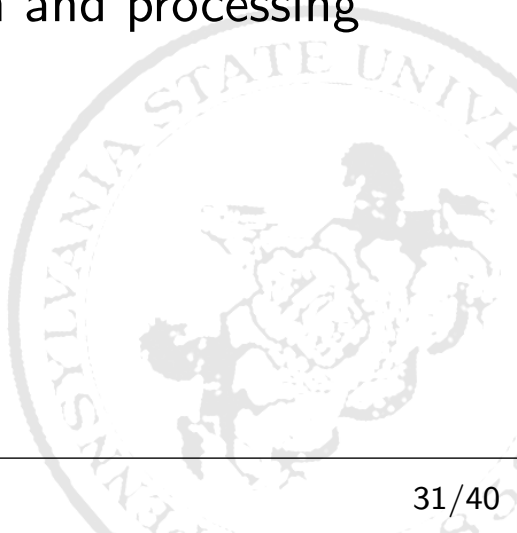
6 Future Work

7 Summary



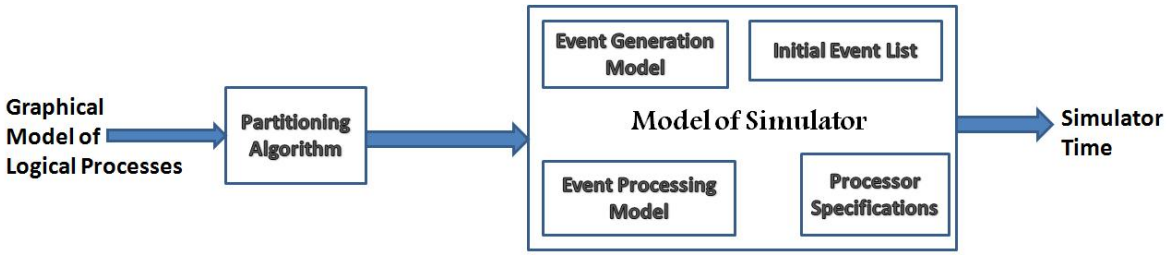
Software Platform to Mimic Distributed Simulator

- The motivating factors:
  1. To abstract the simulation model across different scenarios as an event generation model defined by initial event list and the cause-effect relationships between different events.
  2. To abstract the hardware so as to test with different numbers and speeds of machines.
  3. To make the process independent of any parallel simulator software, thus saving the time needed for understanding finer details of any parallel simulator software tool.
  4. To focus our attention on the simulation time and the synchronization overhead that depend largely on the event generation and processing model only.
- Develop the software network simulator based on NetLogo.



Software Platform to Mimic Distributed Simulator

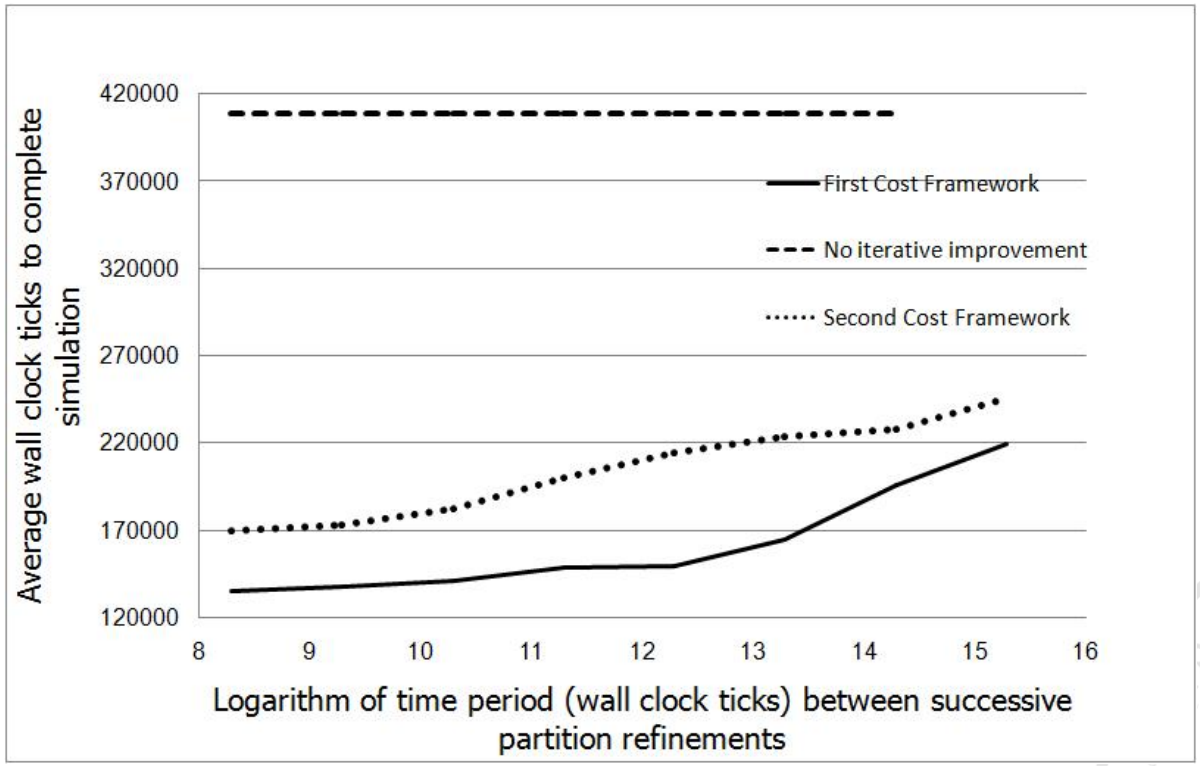
- The simulation of a communication scenario can be described in terms of an event generation model using some parameters:
  1. Machine Specs: Number of machines, speeds, latency of inter machine communications.
  2. Event generation model: Categories of events, how one event spawns another event.
  3. Event Processing Model: Time needed to process each event, local variables affected.
  4. Initial Event List: Events initialized before start of simsulation.



Event Generation Model

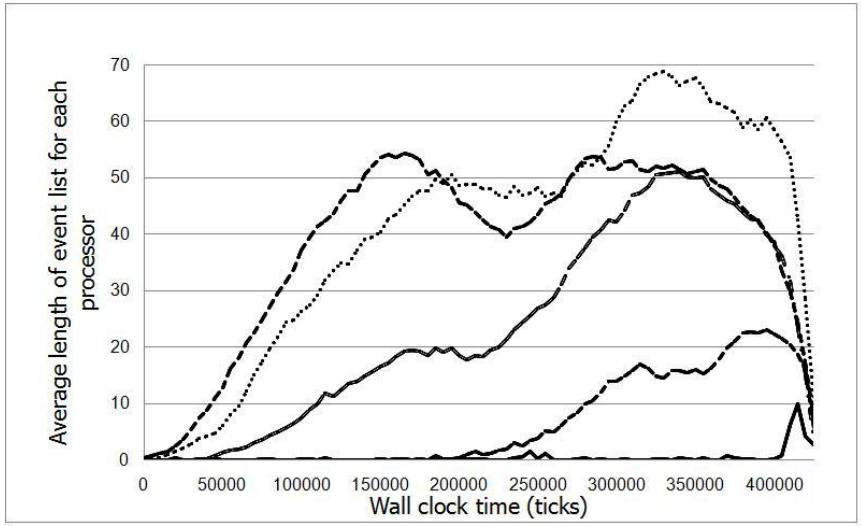
- Optimistic synchronization method: Non causality and roll back. Local and global time.
- Each node represented by an LP.
- Decide the exact scenario: say limited scope flooding.
- The metric to measure performance will be the wall-clock time required to complete the simulation.
- Simulation is complete when the event lists of all the nodes are empty.
- Random graph of 230 LPs is divided among 5 machines. Random node degree of mean 3.
- Normalized speed of machines: $0.1, 0.2, 0.3, 0.3, 0.1$.
- Random initial event list with random time stamp for each initial event between 1 and 10.
- Node weight will be equal to the size of the event list where each event is weighed by the execution time of its associated function.
- The weight of bidirectional edge (i, j) will be equal to the number of events in the event list of (j) that spawn events in (i) plus the converse way.

Simulation time and frequency of refinement

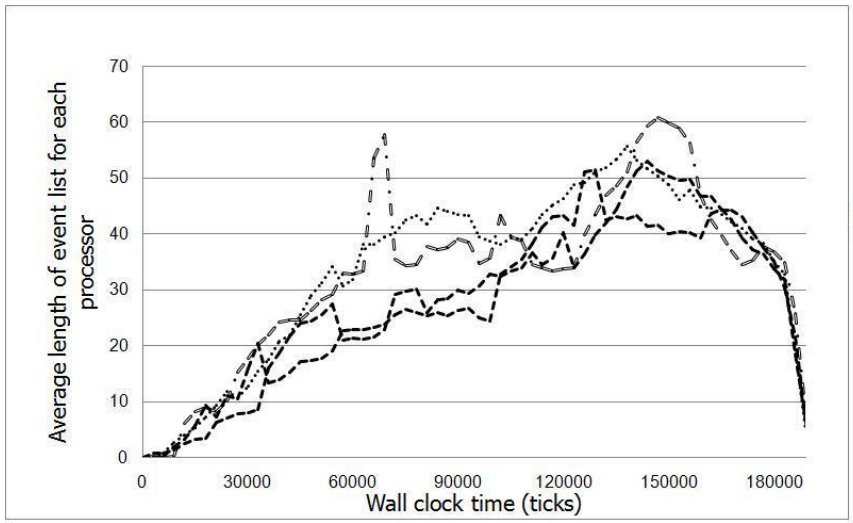


Iterative improvements using first cost framework perform better.

Load distribution across machines with and without refinement



No Iterative Refinement
After Initial Partitioning



Iterative Refinement
After Every 5000 ticks
of Wall clock Time.
Load is More
Balanced

Outline

1 Introduction

2 Background

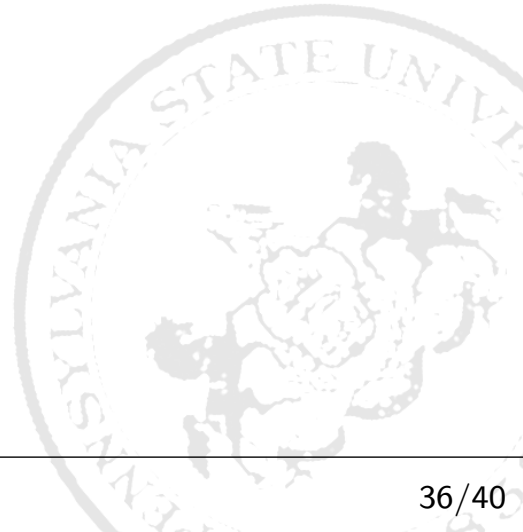
3 Problem Setup

4 Iterative Refinement Steps

5 Software Simulator Platform

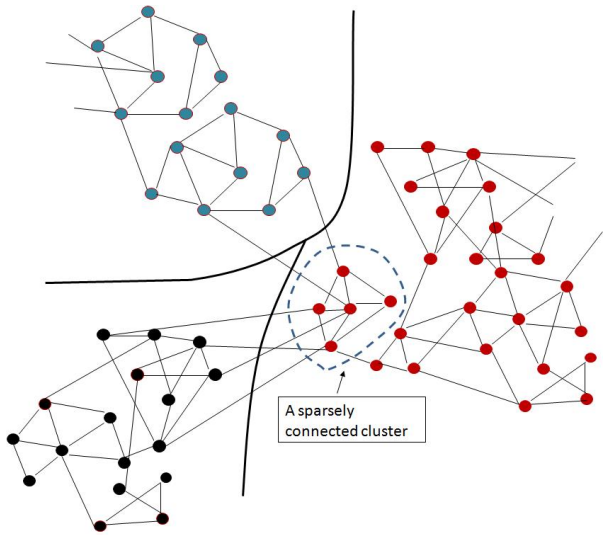
6 Future Work

7 Summary



Overcoming Poor Local Minima

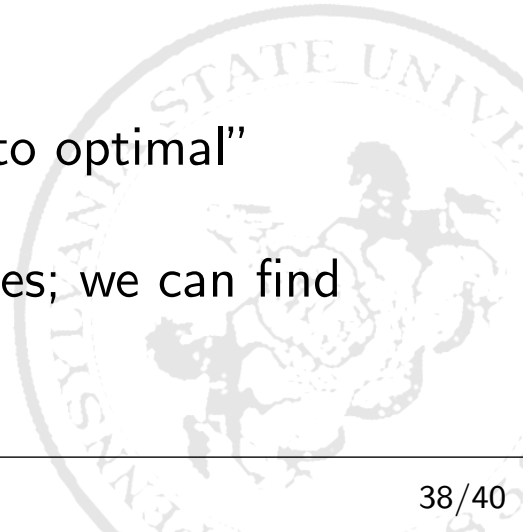
- Inherent non convexity of the cost function might lead to Nash equilibria that correspond to suboptimal partitions.
- Consider iterative cluster-by-cluster transfer instead of node-by-node transfer.
- Increase the breadth of the search space by also considering *sets* of nodes eligible for exchange.
- Consider a connected cluster as a single aggregate node.



- Search space for the optimal strategy now scales exponentially with the number of nodes.
- Choose intelligently the clusters, use the contraction algorithm [Karger 1993].

Overcoming Poor Local Minima

- Distributed Simulated Annealing
  - Simulated annealing [Kirkpatrick 1983] is a classical method of dealing with local minima in discrete optimization problems.
  - One can make moves with non-zero probability that increase the cost function.
  - Allows it get out of local minimum solutions and hence there is good probability of finding better solutions.
  - Studies for graph partitioning have shown to improve cost by approximately 5%.
  - Our case different: Given the distributed nature of our algorithm and the node-by-node transfer based on a certain game formulation.
- Cooperative Game and Pareto Optimality
  - When machines cooperate with each other: A "Pareto optimal" solution.
  - A "semi-cooperative" game: machines exchange nodes; we can find "pairwise stability" in such a case.



Outline

1 Introduction

2 Background

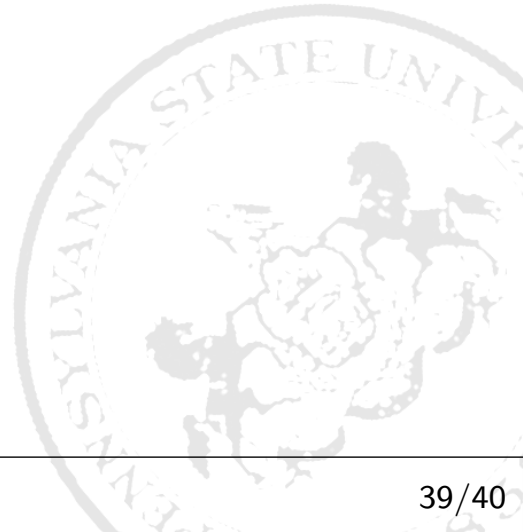
3 Problem Setup

4 Iterative Refinement Steps

5 Software Simulator Platform

6 Future Work

7 Summary



Summary

- We proposed a decentralized load balancing algorithm that iteratively improves the partition based on a game of nodes as players.
- We compare two cost frameworks with different global objectives that approximate the actual simulation time.
- Two corresponding local objectives at node level are defined resulting in decreasing their corresponding global objectives (Lyapunov).
- Local approach is scalable because only partition-level aggregate information is exchanged (rather than node/LP-level).
- We evaluated the two partitioning schemes on a software based simulator model of an optimistic parallel discrete event based simulator.