A Graph Partitioning Game for Distributed Simulation of Networks

A. Kurve[2], C. Griffin[3] and G. Kesidis[1,2]
[1]CS&E Dept, [2]EE Dept and [3]ARL, The Pennsylvania State University, University Park, PA, 16802
{ack205, cxg286, gik2}@psu.edu

***Abstract*—Distributed simulation of large and complex networks requires equitable partitioning of the network model. We propose an iterative partitioning scheme for distributed network simulation that is based on a game theoretic model of simulated network nodes acting as players. We first model the cost function for the nodes and prove that a Nash equilibrium exists for the non-cooperative game in pure strategies. We then propose an iterative algorithm based on our model and prove that it converges to one of the many Nash equilibria. We show with the help of simulations that such a design of cost function moves the system towards optimum partition of the network model as the algorithm converges.**

keywords: parallel network simulation, game theory, graph partitioning, Nash equilibrium.

I. Introduction

Simulation of large communication networks of independent nodes on distributed processors relies on efficient load balancing techniques to overcome loss of efficiency due to synchronization between the processors. The performance of the combined multi-processor system is directly related to the load on the maximally loaded processor, which prevents the system from running any faster. Hence the goal of a load balancing algorithm could be to minimize the load on the maximally loaded processor [14]. This has to be done taking into consideration the performance cost of communicating between the processors. For example, if two routers in a network model under simulation are supposed to generate heavy traffic between them, then the logical processes in the simulation task representing the two routers need to communicate with each other frequently. Hence, it makes sense to have the two logical processes running on the same processor. The performance cost of communication between processors can be large if the processors are remotely connected by, say, Ethernet. In other words, simulation software must suitably divide its logical processes between processors and at the same time minimize the interdependency of threads running on different processors to prevent the extra computational burden caused by rollbacks and deadlock resolution.

The computational burden of each logical process along with the interdependency can be represented by a graph. The nodes in the graph represent logical processes and node weights represent their estimated computation cost. The edges in the graph represent the inter-dependency and the edge weight the amount of communication between the logical processes. It would also make sense to allocate all logical processes associated with a node (router) in a network model under simulation to the same processor. Hence instead of partitioning the graph of logical processes, we can partition the network model itself. To dynamically calculate the node and the edge weights we can use an event generation list to check for future events in line for simulation [1]. We can also use the data generated using pre-simulation runs to estimate the node and the edge weights [12]. Using this information we can calculate the approximate load generated by all the logical processes associated with a node and also the inter node communication traffic.

The classical graph partitioning problem which is known to be NP-complete [6] can be heuristically solved using one of the many techniques such as those discussed in [7]. Most of the proposed techniques are fully centralized. Moreover, due to the dynamic event-driven nature of the node and edge weights, graphs in our proposed problem are highly dynamic. The dynamic and unpredictable nature of nodes in the network model suggest the need for a self correcting and distributed graph partitioning algorithm. Game theoretic tools provide insights into the design of distributed algorithms with competition. In this paper, a game theoretic formulation of the classical load balancing problem is studied and this formulation is extended to include the inter-processor communication constraints. We prove that a Nash equilibrium exists in pure strategies under a specific network partition cost function. We then design an iterative graph partitioning algorithm where each partition takes turns to give up its "most dissatisfied" node. The "most dissatisfied" node is the one which would be able to decrease its cost the most by switching processors. We show that this algorithm converges to a Nash equilibrium. We also propose a coarse initial partition first proposed in [10] that acts as a starting point for the iterative algorithm. We argue that initial partitioning need not be precise owing to the fact that the network model parameters *i.e.*, node and edge weights, are difficult to estimate *a priori* and also change with time. Hence we rely on distributed iterative partition refinement steps based on our game theoretic model to improve the partition and track the changes in the network model under simulation.

II. Literature Survey

The resource intensive nature of simulation of large networks has prompted researchers to look for solution either in scaled down topologies or distributed simulation on parallel

This work was funded in part by NSF Trustworthy Computing grant 0831068.

978-0-9836283-1-6

This paper was peer reviewed by subject matter experts for publication in the Proceedings of Cnet 2011

processors. Papers on scaled down network topologies such as [18] aim to find better techniques to omit the intricate details of a large network while at the same time retaining the accuracy of the simulation. The partitioning problem, which naturally accompanies any multiprocessor implementation, has been looked at from different perspectives: algorithm design [8] , benchmarking [21] and implementation [3]. [11] studies partitioning algorithms for distributed virtual environments (DVEs) using a divide and conquer technique. The divide and conquer technique divides the avatars into regions of influence and the initial partitioning is performed on these regions (group of avatars). [21] focuses mainly on the characterization of factors that affect the simulation time called "benchmarking". The benchmarking technique customizes the partition model to a specific network model. A major issue in network model partitioning for simulations is the lack of knowledge about the computational load generated by nodes and the inter node traffic during compilation of the network model. Hence, assuming unit computational load for each node (node weight) and unit inter node communication (edge weight), a simple and attractive solution can be obtained [17]. Dealing with highly dynamic networks adds complexity as it implies fast changing node and edge weights. In this case an optimal partitioning fails to remain so at later time instances. Regularly performing a total refresh of the network is not a computationally viable option. [16] - [20] investigate the problem of dynamic load balancing or graph partitioning. [3] focuses on the implementation details of partitioning and measuring parameters needed for efficient partitioning. [17] proposes initial partitioning based on equal node and edge weights and the partitioning is refined repeatedly via a load balancing cycle and a communication refinement cycle. [16] uses the fact that simulations consist of multiple runs with same parameter settings and uses MeTiS software [8] for initial partitioning.

An extensive game theoretic study of the load balancing problem has been done in [4]. Chapter 20 of [15] titled "Selfish Load Balancing" gives a nice overview of results in this area. In this paper, we look at a similar formulation of the problem, but we also consider the inter-processor communication cost. Hence we can claim that our study has a more generalized objective than load balancing. We also use insights gained from our theoretical results to design an iterative partitioning scheme to deal with dynamically changing load. The work presented in [12] is by far the closest to the work presented in this paper with some major differences. [12] proposed an iterative method based on the "gain" of each node, however, the gain only minimizes the cost of the cut in the graph. Moreover, the "forceful" convergence occurs by allowing the nodes to migrate only once. In our work convergence is spontaneous and a natural outcome with a guaranteed partition refinement. We support our claims with a theoretical study of the problem. Game theoretic study for the clustering problem has been done in works such as [2] where evolutionary game theory has been applied to the similarity based clustering problem. The equilibrium in terms of the hypothesis of individual players pertaining to cluster memberships is reached by unsupervised method of learning from mistakes in a large population of players. Our method instead uses the classical notion of game theory and Nash equilibrium under strict competition.

III. Problem Setup and Outline

Consider a graph $G = (V, E)$ representing the network model to be simulated. First, we first must estimate the computational load generated by each node and the communication load between the logical processes associated with each node to assign node weights and edge weights, respectively, of the graph. Second, we wish to find a distributed technique to equitably load-balance among the processors that also takes into consideration the inter-processor communication cost. We assume that the simulation program consists of a mechanism that estimates the node weights (b_i) and the edge weights (c_{ij}) from the list of future events. We focus on the second part which deals with finding an optimum partition. The graph G is partitioned between K processors. Let b_i represent the computational load of the i^{th} node. Let c_{ij} denote the cost of communicating over the edge (i, j) which represents the average traffic between node i and node j.

Most techniques proposed in the area of network simulation on distributed processors use one of the many graph partitioning heuristics based on multilevel partitioning [7]. The multilevel partitioning algorithm was initially developed for centralized implementation and later modified to exploit the parallelism of a collection of processors. The centralized scheme requires a dedicated task which can run either on one of the K processors or on a separate dedicated host machine. As such, there is a need for exploring a graph partitioning heuristic that naturally exploits the parallelism of the processors with minimal centralized assistance to coordinate the algorithm. We wish to distribute among the participating hosts as much as possible the load of tracking our partition as the graph characteristics changes. We assume a connected network model graph, otherwise we can easily convert a disconnected graph into a connected one by adding edges of weight zero between the disconnected subgraphs. Hence the initial partitioning [10] consists of finding focal nodes and then expanding partitions hop-by-hop (similar to a flooding algorithm). The initial partition is sub-optimal and so is refined using the iterative refinement schemes.

Consider a centralized formulation of the graph partitioning problem. Let $x_{ki} = 1$ if node i belongs to processor k; otherwise $x_{ki} = 0$. We require $\sum_k x_{ki} = 1 \quad \forall i = 1, 2, ..., |V|$. Let w_k be the normalized capacity of the k^{th} processor so that $\sum_k w_k = 1$. The centralized 'K'-way partitioning problem aims to solve the following integer program.

$$\min \quad C_0 = \sum_{k=0}^{K} \left(\frac{\sum_{j \in V} x_{kj} b_j}{w_k} - \sum_j b_j \right)^2 + \frac{\mu}{2} \sum_{i,j} c_{ij} x_{ki} (1 - x_{kj}) \qquad (1)$$

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

Here μ denotes the relative weight given to the inter-host communication cost. This is a quadratic integer programming

problem the convexity of which depends on the network graph. In most cases it will not be a convex problem. We can think of decomposing this problem into a set of K subproblems each of which is solved by a single partition. However, the constraints $\sum_k x_{kj} = 1 \ \forall \ j$ make such a decomposition difficult to realize.

IV. A Modified Partitioning Game

A natural way of formulating a game for graph partitioning would consist of partitions/processors as players and each partition bidding for the nodes that help to maximize its utility/payoff or minimize its cost. This can be formulated as a combinatorial auction game. Combinatorial auctions [5] have been studied widely, however we are not aware of any work that uses them for graph partitioning. Treating the nodes as players who may choose from among partitions (processor) to minimize their own costs is an alternative approach to formulating the partitioning problem as a game. This approach avoids the decomposition problem associated with the constraints $\sum_k x_{kj} = 1 \ \forall \ j$. We try to find a suitable cost function that guarantees a Nash equilibrium for the game and gives us an optimum partition in some sense at the Nash equilibrium.

A. Node Cost Function

Suppose there are N nodes and K partitions. Let $r_i \in \{1, 2, ...K\}$ be the partition chosen by the i^{th} node. Let the the normalized capacity or speed of the i^{th} processor be:

$$w_i = \frac{s_i}{\sum_{j=1}^{K} s_j},$$

where s_i is the speed of i^{th} processor. Let us assign the following cost function to the i^{th} node.

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

where the computational cost of a node intuitively depends on two factors: the existing load on the assigned processor:

$$\sum_{j:r_j=r_i, j\neq i} b_j$$

and the computational load that the node will bring to the processor (b_i). Suppose that the computational load generated by the node is zero ($b_i = 0$), then the computational part of the cost should be zero. Hence multiplication by b_i makes sense. The second term in the sum represents the weight of edges that connect the i^{th} node with nodes in other partitions, which will incentivize the node to choose a partition with which it is well connected.

B. Nash Equilibrium for the modified game

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

$$C_i(r_i^*, r_{-i}^*) \leq C_i(r_i, r_{-i}^*) \quad \forall r_i \in \{1, 2, ..., K\} \ \forall i. \tag{3}$$

Theorem 1: For the game described above, a Nash Equilibrium exists in pure strategies.

Proof: To construct a proof of Theorem 1, consider the following combinatorial optimization problem:

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

Hence, $\Psi(\mathbf{r})$ is the sum of the cost of all nodes for a particular assignment. We can interpret it as the social welfare function. We can find an assignment $\hat{\mathbf{r}} = (\hat{r}_1, \hat{r}_2, ..., \hat{r}_N)$ which minimizes the objective. We prove, by contradiction, that $\hat{\mathbf{r}}$ is also the Nash equilibrium for our game.

Suppose node l is moved from processor k_1 to processor k_2, then we can divide the objective function as follows:

$$\begin{aligned}
\Psi(\mathbf{r}) = &\left(\frac{b_l}{w_{r_l}} \sum_{\substack{j:r_j=r_l \\ j\neq l}} b_j + \frac{\mu}{2} \sum_{j:r_j\neq r_l} c_{lj} \right) \\
&+ \sum_{\substack{i:i\neq l \\ r_i=k_1}} \left(\frac{b_i}{w_{r_i}} \sum_{\substack{j:r_j=r_i \\ j\neq i}} b_j + \frac{\mu}{2} \sum_{j:r_j\neq r_i} c_{ij} \right) \\
&+ \sum_{\substack{i:i\neq l \\ r_i=k_2}} \left(\frac{b_i}{w_{r_i}} \sum_{\substack{j:r_j=r_i \\ j\neq i}} b_j + \frac{\mu}{2} \sum_{j:r_j\neq r_i} c_{ij} \right) \\
&+ \sum_{\substack{i:i\neq l, r_i\neq k_1 \\ r_i\neq k_2}} \left(\frac{b_i}{w_{r_i}} \sum_{\substack{j:r_j=r_i \\ j\neq i}} b_j + \frac{\mu}{2} \sum_{j:r_j\neq r_i} c_{ij} \right)
\end{aligned}$$

Hence

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

The first term is the cost of node l, the second term is sum of the costs of the nodes that are assigned to the same processor as l, the third term is the sum of the costs of the nodes that belong to the prospective processor of l and the fourth term is the sum of the costs of the processors that belong to neither the current processor of node l nor the prospective processor of node l.

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 $\hat{r}_l$ processor to r_l^* processor. Then the new assignment vector is $\mathbf{r}^* = (r_1^*, r_2^*, ..., r_N^*)$ and $C_l(\mathbf{r}^*) - C_l(\hat{\mathbf{r}}) < 0$:

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

Claim 1: $\Psi(\mathbf{r}^*|i = l) - \Psi(\hat{\mathbf{r}}|i = l) < 0$

Proof: The proof of this claim is trivial. We know that,

$$\Psi(\mathbf{r}^*|i = l) - \Psi(\hat{\mathbf{r}}|i = l) = C_l(\mathbf{r}^*) - C_l(\hat{\mathbf{r}}),$$

and we know from our assumption that

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

■

Claim 2:

$$\Psi(\mathbf{r}^*|i \neq l, r_i^* \neq \hat{r}_l, r_i^* \neq r_l^*) - \Psi(\hat{\mathbf{r}}|i \neq l, \hat{r}_i = \hat{r}_l, \hat{r}_i \neq r_l^*) = 0$$

Proof: This term represents the sum of the change in the cost values of nodes which belong to neither of the two processors involved in the transfer of node l. Hence there is no change in their individual cost function. ■

Claim 3:

$$\Psi(\mathbf{r}^*|i \neq l, r_i^* = \hat{r}_l) - \Psi(\hat{\mathbf{r}}|i \neq l, \hat{r}_i = \hat{r}_l) + \Psi(\mathbf{r}^*|i \neq l, r_i^* = r_l^*) - \Psi(\hat{\mathbf{r}}|i \neq l, \hat{r}_i = r_l^*) < 0$$

Proof:

$$\Psi(\mathbf{r}^*|i \neq l, r_i^* = \hat{r}_l) - \Psi(\hat{\mathbf{r}}|i \neq l, \hat{r}_i = \hat{r}_l) = \sum_{\substack{i:i\neq l \\ r_i^*=\hat{r}_l}} \left(\frac{b_i}{w_{r_i^*}} \sum_{\substack{j:r_j^*=r_i^* \\ j\neq i}} b_j + \frac{\mu}{2} \sum_{j:r_j^*\neq r_i^*} c_{ij} \right) = \sum_{\substack{i:i\neq l \\ \hat{r}_i=\hat{r}_l}} \left(\frac{b_i}{w_{\hat{r}_i}} \sum_{\substack{j:\hat{r}_j=\hat{r}_i \\ j\neq i}} b_j + \frac{\mu}{2} \sum_{j:\hat{r}_j\neq\hat{r}_i} c_{ij} \right) = \sum_{\substack{i:i\neq l \\ r_i^*=\hat{r}_l}} \left(\frac{b_i}{w_{\hat{r}_l}} \sum_{\substack{j:r_j^*=\hat{r}_l \\ j\neq i}} b_j + \frac{\mu}{2} \sum_{j:r_j^*\neq\hat{r}_l} c_{ij} \right) - \sum_{\substack{i:i\neq l \\ \hat{r}_i=\hat{r}_l}} \left(\frac{b_i}{w_{\hat{r}_l}} \sum_{\substack{j:\hat{r}_j=\hat{r}_l \\ j\neq i}} b_j + \frac{\mu}{2} \sum_{j:\hat{r}_j\neq\hat{r}_l} c_{ij} \right).$$

We know that the two sets $\{i : i \neq l, \hat{r}_i = \hat{r}_l\}$ and $\{i : i \neq l, r_i^* = \hat{r}_l\}$ are equal. This is so because all other assignments except r_l are the same in the new assignment vector $\mathbf{r}^*$. So,

$$\Psi(\mathbf{r}^*|i \neq l, r_i^* = \hat{r}_l) - \Psi(\hat{\mathbf{r}}|i \neq l, \hat{r}_i = \hat{r}_l) = \sum_{\substack{i:i\neq l \\ r_i^*=\hat{r}_l}} \left(\frac{b_i}{w_{\hat{r}_l}} \sum_{\substack{j:r_j^*=\hat{r}_l \\ j\neq i}} b_j - \frac{b_i}{w_{\hat{r}_l}} \sum_{\substack{j:\hat{r}_j=\hat{r}_l \\ j\neq i}} b_j + \frac{\mu}{2} \sum_{j:r_j^*\neq\hat{r}_l} c_{ij} - \frac{\mu}{2} \sum_{j:\hat{r}_j\neq\hat{r}_l} c_{ij} \right) = \sum_{\substack{i:i\neq l \\ r_i^*=\hat{r}_l}} \left(\frac{b_i}{w_{\hat{r}_l}} \Big(\sum_{\substack{j:r_j^*=\hat{r}_l \\ j\neq i}} b_j - \sum_{\substack{j:\hat{r}_j=\hat{r}_l \\ j\neq i}} b_j \Big) + \frac{\mu}{2} \Big(\sum_{j:r_j^*\neq\hat{r}_l} c_{ij} - \sum_{j:\hat{r}_j\neq\hat{r}_l} c_{ij} \Big) \right)$$

We know that for $i \neq l$, $\{j : r_j^* = \hat{r}_l, j \neq i\}$ represents the set of nodes except the l^{th} and i^{th} node which are currently assigned to $\hat{r}_l{}^{th}$ processor and $\{j : \hat{r}_j = \hat{r}_l, j \neq i\}$ represents the set of nodes except the l^{th} and i^{th} node whose new assignment is the $\hat{r}_l{}^{th}$ processor. So, for $i \neq l$, $\{j : r_j^* = \hat{r}_l, j \neq i\} \setminus \{j : \hat{r}_j = \hat{r}_l, j \neq i\} = l$. This implies

$$\sum_{\substack{j:r_j^*=\hat{r}_l \\ j\neq i}} b_j - \sum_{\substack{j:\hat{r}_j=\hat{r}_l \\ j\neq i}} b_j = -b_l.$$

Also, $\{j : r_j^* \neq \hat{r}_l\} \setminus \{j : \hat{r}_j \neq \hat{r}_l\} = l$. So, we can write

$$\sum_{j:r_j^*\neq\hat{r}_l} c_{ij} - \sum_{j:\hat{r}_j\neq\hat{r}_l} c_{ij} = c_{il},$$

which implies,

$$\Psi(\mathbf{r}^*|i \neq l, r_i^* = \hat{r}_l) - \Psi(\hat{\mathbf{r}}|i \neq l, \hat{r}_i = \hat{r}_l) = \sum_{\substack{i:i\neq l \\ r_i^*=\hat{r}_l}} \left(-b_l \Big(\frac{b_i}{w_{\hat{r}_l}} \Big) + \frac{\mu}{2} c_{il} \right).$$

Using a similar approach, we can prove:

$$\Psi(\mathbf{r}^*|i \neq l, r_i^* = r_l^*) - \Psi(\hat{\mathbf{r}}|i \neq l, \hat{r}_i = r_l^*) = \sum_{\substack{i:i\neq l \\ r_i^*=r_l^*}} \left(b_l \Big(\frac{b_i}{w_{r_l^*}} \Big) - \frac{\mu}{2} c_{il} \right).$$

So,

$$\Psi(\mathbf{r}^*|i \neq l, r_i^* = \hat{r}_l) - \Psi(\hat{\mathbf{r}}|i \neq l, \hat{r}_i = \hat{r}_l) + \Psi(\mathbf{r}^*|i \neq l, r_i^* = r_l^*) - \Psi(\hat{\mathbf{r}}|i \neq l, \hat{r}_i = r_l^*) = \sum_{\substack{i:i\neq l \\ r_i^*=\hat{r}_l}} \left(-b_l \Big(\frac{b_i}{w_{\hat{r}_l}} \Big) + \frac{\mu}{2} c_{il} \right) + \sum_{\substack{i:i\neq l \\ r_i^*=r_l^*}} \left(b_l \Big(\frac{b_i}{w_{r_l^*}} \Big) - \frac{\mu}{2} c_{il} \right).$$

Consider

$$C_l(\mathbf{r}^*) - C_l(\hat{\mathbf{r}}) = b_l \sum_{\substack{i:r_i^*=r_l^* \\ i\neq l}} \frac{b_i}{w_{r_l^*}} + \frac{\mu}{2} \sum_{\substack{i:r_i^*\neq r_l^* \\ i\neq l}} c_{il} - b_l \sum_{\substack{i:\hat{r}_i=\hat{r}_l \\ i\neq l}} \frac{b_i}{w_{\hat{r}_l}} - \frac{\mu}{2} \sum_{\substack{i:\hat{r}_i\neq\hat{r}_l \\ i\neq l}} c_{il}.$$

Now,

$$\sum_{\substack{i:r_i^*\neq r_l^* \\ i\neq l}} c_{il} - \sum_{\substack{i:\hat{r}_i\neq\hat{r}_l \\ i\neq l}} c_{il} = \sum_{\substack{r_i^*=\hat{r}_l \\ i\neq l}} c_{il} + \sum_{\substack{i:r_i^*\neq r_l^* \\ r_i^*\neq\hat{r}_l \\ i\neq l}} c_{il} - \sum_{\substack{\hat{r}_i=r_l^* \\ i\neq l}} c_{il} - \sum_{\substack{i:\hat{r}_i\neq\hat{r}_l \\ \hat{r}_i\neq r_l^* \\ i\neq l}} c_{il}.$$

But,

$$\sum_{\substack{i:r_i^*\neq r_l^* \\ r_i^*\neq\hat{r}_l \\ i\neq l}} c_{il} = \sum_{\substack{i:\hat{r}_i\neq\hat{r}_l \\ \hat{r}_i\neq r_l^* \\ i\neq l}} c_{il}.$$

Hence,

$$\sum_{\substack{i:r_i^* \neq r_l^* \\ i \neq l}} c_{il} - \sum_{\substack{i:\hat{r}_i \neq \hat{r}_l \\ i \neq l}} c_{il} = \sum_{\substack{r_i^* = \hat{r}_l \\ i \neq l}} c_{il} - \sum_{\substack{\hat{r}_i = r_l^* \\ i \neq l}} c_{il}$$

$$\Rightarrow C_l(\mathbf{r}^*) - C_l(\hat{\mathbf{r}}) = -b_l \sum_{\substack{i:\hat{r}_i = \hat{r}_l \\ i \neq l}} \frac{b_i}{w_{\hat{r}_l}} + \frac{\mu}{2} \sum_{\substack{r_i^* = \hat{r}_l \\ i \neq l}} c_{il} + b_l \sum_{\substack{i:r_i^* = r_l^* \\ i \neq l}} \frac{b_i}{w_{r_l^*}} - \frac{\mu}{2} \sum_{\substack{\hat{r}_i = r_l^* \\ i \neq l}} c_{il}.$$

But we know that $\{i : \hat{r}_i = \hat{r}_l, i \neq l\} = \{i : r_i^* = \hat{r}_l, i \neq l\}$ and $\{\hat{r}_i = r_l^*, i \neq l\} = \{r_i^* = r_l^*, i \neq l\}$. So,

$$C_l(\mathbf{r}^*) - C_l(\hat{\mathbf{r}}) = - b_l \sum_{\substack{i:r_i^* = \hat{r}_l \\ i \neq l}} \frac{b_i}{w_{\hat{r}_l}} + \frac{\mu}{2} \sum_{\substack{r_i^* = \hat{r}_l \\ i \neq l}} c_{il} + b_l \sum_{\substack{i:r_i^* = r_l^* \\ i \neq l}} \frac{b_i}{w_{r_l^*}} - \frac{\mu}{2} \sum_{\substack{r_i^* = r_l^* \\ i \neq l}} c_{il} =$$

$$\sum_{\substack{i:i \neq l \\ r_i^* = \hat{r}_l}} \left(-b_l \Big(\frac{b_i}{w_{\hat{r}_l}} \Big) + \frac{\mu}{2} c_{il} \right) + \sum_{\substack{i:i \neq l \\ r_i^* = r_l^*}} \left(b_l \Big(\frac{b_i}{w_{r_l^*}} \Big) - \frac{\mu}{2} c_{il} \right) =$$

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

Thus it follows that:

$$\Psi(\mathbf{r}^* | i \neq l, r_i^* = \hat{r}_l) - \Psi(\hat{\mathbf{r}} | i \neq l, \hat{r}_i = \hat{r}_l) + \Psi(\mathbf{r}^* | i \neq l, r_i^* = r_l^*) - \Psi(\hat{\mathbf{r}} | i \neq l, \hat{r}_i = r_l^*) < 0$$

This completes the proof of this claim. ■

Hence using Claims 1, 2 and 3, $\Psi(\mathbf{r}^*) - \Psi(\hat{\mathbf{r}}) < 0$. This contradicts our assumption that $\hat{\mathbf{r}}$ is an optimal solution of our problem. Hence $\hat{\mathbf{r}}$ is also the Nash equilibrium. ■

V. Algorithm Design and Analysis

A. Initial Partitioning

The goal of an initial partitioning is not to obtain an optimum partition, but an initial distributed node-to-host assignment from which the iterative partitioning can commence potentially after the simulation actually begins. We argue that due to initial uncertainty of node and edge weights and their dynamic nature, an optimum partition is not possible *a priori*. As a result, we employ a simple initial partitioning method in which each processor chooses initial "focal nodes" from among the nodes of the graph and then expands hop-by-hop to include neighboring nodes. To avoid contention between partitions we require that each processor wait a random amount of time after every hop and check for a semaphore before claiming ownership of new nodes. Unit edge and node weights are assumed during initial partitioning. The choice of the focal nodes is important to ensure a high probability of a good initial partition. We discuss this problem briefly in the appendix. As will be described later, the iterative partition refinement algorithm converges to one of the local minima and hence a good initial partition might improve the chances of converging to a global minimum.

B. Iterative Partition Refinement

The refinement algorithm demonstrates the practical utility of the game theoretic framework described previously. During the refinement step, each processor takes turns to decide the "most dissatisfied" node in its partition and transfers its ownership to a new processor. The "most dissatisfied" node is the one which would benefit the most in terms of its cost by changing its processor. The cost function used here is the same as given in equation 2. Each partition maintains a list of nodes currently owned by it along with their current costs and costs if they were to change processor for each processor. Hence $C_i(k)$ represents the cost of the i^{th} node if it were to be assigned to the k^{th} processor and $C_i(r_i)$ represents the current cost of the i^{th} node. We define the dissatisfaction of the i^{th} node as

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

Hence the most dissatisfied node in a partition is the one with maximum value of $\Im$. If $\Im = 0$ for the most dissatisfied node, then the partition forsakes its turn. When all the partitions have most dissatisfied nodes with $\Im = 0$ then the algorithm has converged to a local minimum as will be proved later.

Algorithm 1 Iterative Partition Refinement Algorithm

```
1:  repeat
2:    repeat
3:      Wait
4:    until trigger is received
5:    if ReceiveNodeTrigger then
6:      Add the new node to the list
7:      Update cost functions and recalculate ℑ for all nodes
        in my partition
8:    if RegularUpdateTrigger then
9:      Update cost functions for the new assignment and
        recalculate ℑ for all nodes in my partition.
10:   if TakeMyTurnTrigger then
11:     Transfer the "most dissatisfied" node to the suitable
        processor
12:     Send ReceiveNodeTrigger to the destination proces-
        sor
13:     Send RegularUpdateTrigger to all other processors.
14:     Send TakeMyTurnTrigger to the next processor
15:     Update cost functions for the new assignment and
        recalculate ℑ for all nodes in my partition.
16: until forever
```

C. Convergence of Algorithm

Theorem 2: The iterative partition refinement algorithm converges.

Proof: $\Psi(\mathbf{r})$ defined in Theorem 1 will be shown to be a *potential* function. Suppose the l^{th} node is transferred from its current allocated processor r_l to a new processor r_l^*. Call the new assignment vector $\mathbf{r}^*$. Then obviously,

$$C_l(\mathbf{r}^*) - C_l(\mathbf{r}) < 0. \tag{4}$$

From Theorem 1 we also know that

$$\begin{aligned}
\Psi(\mathbf{r}^*) - \Psi(\mathbf{r}) = \Psi(\mathbf{r}^*|i=l) - \Psi(\mathbf{r}|i=l) + \\
\Psi(\mathbf{r}^*|i\neq l, r_i^*=r_l) - \Psi(\mathbf{r}|i\neq l, r_i=r_l) + \\
\Psi(\mathbf{r}^*|i\neq l, r_i^*=r_l^*) - \Psi(\mathbf{r}|i\neq l, r_i=r_l^*) + \\
\Psi(\mathbf{r}^*|i\neq l, r_i^*\neq r_l, r_i^*\neq r_l^*) - \Psi(\mathbf{r}|i\neq l, r_i=r_l, r_i\neq r_l^*) = \\
[\Psi(\mathbf{r}^*|i=l) - \Psi(\mathbf{r}|i=l)] + \\
[\Psi(\mathbf{r}^*|i\neq l, r_i^*=r_l) - \Psi(\mathbf{r}|i\neq l, r_i=r_l) + \\
\Psi(\mathbf{r}^*|i\neq l, r_i^*=r_l^*) - \Psi(\mathbf{r}|i\neq l, r_i=r_l^*)] = \\
[C_l(\mathbf{r}^*) - C_l(\mathbf{r})] + [C_l(\mathbf{r}^*) - C_l(\mathbf{r})] \\
= 2\big(C_l(\mathbf{r}^*) - C_l(\mathbf{r})\big) < 0
\end{aligned}$$

Hence for every iterative step the potential function $\Psi(\mathbf{r})$ decreases and we know that due to bounded nature of the combinatorial optimization problem, there exists an assignment vector $\mathbf{r}$ which will yield the minimum value of Ψ. Hence an achievable lower bound for the potential function exists and so, we can conclude that the algorithm converges. ■

D. Price of Anarchy

The potential function $\Psi(\mathbf{r})$ may not be convex. So, the algorithm can possibly converge to only a local minimum as dictated by the properties of the graph. The algorithm does not guarantee an optimal performance and in some cases the "price of anarchy" (here, the gap between global and local optima) might be significant. As in other problems of similar nature, a meta-heuristic approach such as (distributed) simulated annealing can be used to get a good approximation to the global optimum. Also, restricting our search to the "most dissatisfied node" constrains the strategy space. If we consider coordinated play by a *set* of nodes/players, instead of an individual node, we can push the algorithm out of the only local optimum equilibrium. So, we would like to consider more than one nodes for transfer during a single iteration. However, in this case the search space increases exponentially with the number of nodes. To address this issue, in the future we will explore the use a "sparse cut" criterion to narrow our search for the "most dissatisfied" set of nodes.

E. Algorithm Complexity

Every transfer of a single node necessitates re-computation of the cost function for each node and for each partition. This might seem computationally onerous if we calculate the cost function naively. But we can optimize our computational effort by using the fact that computations can be spatially localized, in the sense that the communication cost of only those nodes will be affected which are connected to the node involved in the transfer. Also, the computational cost of each node can be calculated using a common variable array containing the current aggregate weights of each partition. We can see that the computational effort scales linearly with the number of nodes associated with the processor. To optimize further we can allow simultaneous transfer of nodes by more than one processor if they are distant in the graph and if they are between disjoint pairs of processors. Furthermore, hierarchical search techniques can be employed to find the "most dissatisfied" node and arbitrate the transfer of nodes. As discussed previously, the hierarchy of processors helps to reduce the communication overhead for coordination between the processors.

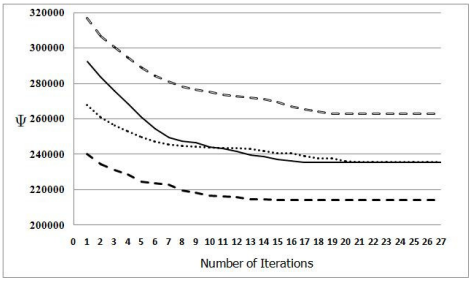

Fig. 1: Potential function versus number of iterations

VI. Simulation Study

In the first phase of experiments, we verified our theoretical claims via a numerical study. We used NetLogo [13] which is a multi-agent simulation software to create random graphs representing the network model under simulation. We randomly generated node and edge weights each with a mean 5. The number of nodes were fixed to 300 and the number of partitions to 5. The degree of each node varied from 3 to 6 randomly. We also fixed the value of μ to 100. Every cycle of iterative refinement steps are preceded by initial partitioning and we used the heuristic described in the appendix to choose the initial focal nodes. Fig 1 shows the decrease in the value of the potential function Ψ as the algorithm iterates. Here an iteration consist of one round of play where each processor takes turns to give up the "most dissatisfied" node in its partition. We see that it ultimately converges to a local minimum. We plot this curve for different initial partitions. Fig 2 shows the decrease in the centralized cost and we see that it decreases with the number of iterations reaffirming our belief that the node cost function does indeed help to minimize social welfare interpreted here as the centralized cost of partitioning given by equation1. Moreover, we observe that the decreasing trend for the quantities shown in the two figures is similar although they represent two disparate quantities.

In the second phase, we tested the partitioning algorithm on a software based model of an asynchronous optimistic discrete-event based simulator. Such a simulator can be described by a deterministic model: if we know the event-generation model, the processor load and speed which gives us the time interval between two instruction cycles, the function

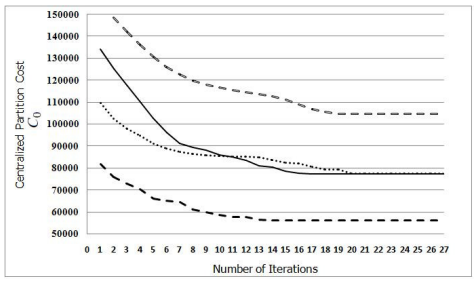

Fig. 2: Centralized Partitioning Cost versus number of iterations

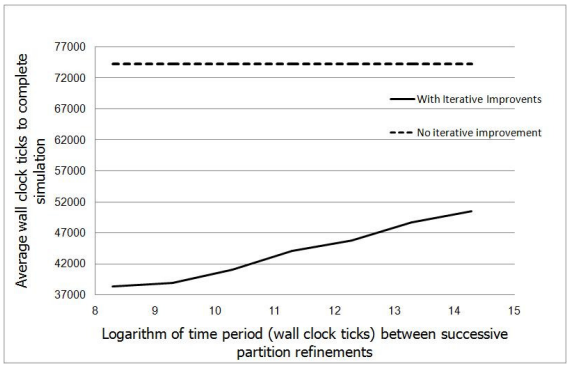

Fig. 3: Simulation time versus partition refinement frequency

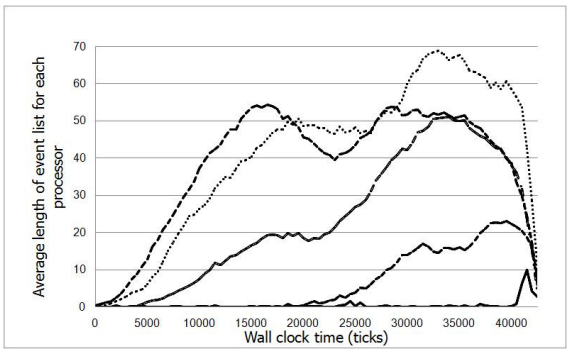

Fig. 4: Variance of load across processors without refinement

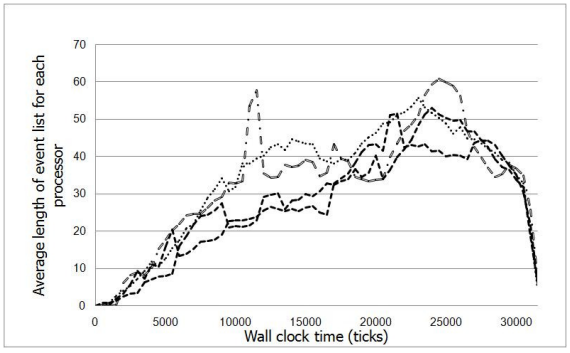

Fig. 5: Variance of load across processors with refinement

that each logical process performs for each event and the number of instruction cycles needed for it, the maximum length of the memory that stores past history needed in case of a roll-back, and the assignment of logical processes to processors (which we will be studying). We are interested in the wall-clock time (in ticks) for the simulation to complete and the number of roll-backs that arise from load imbalance. We study a simple communication scenario consisting of limited-scope flooding sessions. The node weight is equal to the size of the event list, where each event is weighed by the execution time of its associated function and the weight of edge (i, j) is the number of events in the event list of i (j) that spawn events in j (i). Using Netlogo [13], we generated a random graph of 300 nodes depicting the graphical model of the logical processes. We generated a random initial event list for each LP and kept it constant throughout the experiment. The initial partition was also retained and kept constant for all the runs so to compare the final results. For each setting of the partition refinement frequency, average simulation time for 10 runs was calculated and is shown in fig 3. With just the initial partitioning *i.e.*, without any refinement the average simulation time was 74264.9. We observed that the simulation time decreases as we increase the frequency of refinement.

Fig 4 shows the load across all the processors as wall clock time progresses. The processor load is calculated as the average length of event list of LPs assigned to the processor. We observed that this variance is very high suggesting that initial partitioning alone is not sufficient to balance the load. Fig 5 shows the same when we apply iterative partition refinements after every 2500 ticks of wall-clock time.

VII. Future Work

In the future, we plan to carry out more extensive study with event generation modes of varying load generation characteristics. On the theoretical front, we can study the price of anarchy of using such a technique as opposed to centralized partitioning. Also, we will explore the use of meta-heuristics such as simulated annealing to avoid local minima. Local minima are reached because of node-by-node transfer, so we can also think of finding the "most dissatisfied cluster" instead of the "most dissatisfied node" to exchange; however this may be computationally infeasible and so we will need to find methods to intelligently choose "ideal clusters" of nodes suitable for transfer. We will also try pairwise exchange of nodes, where a pair of processors exchanges two "dissatisfied" nodes between themselves. This can be studied as a "semi-cooperative" game.

VIII. Summary and Conclusions

We studied the graph partitioning problem in the context of distributed network simulation. For our goal of a distributed technique of partitioning which works iteratively on dynamic graphs, we studied a game theoretic formulation of the problem and considered each node as a player. The insights gained from this model allowed us to design a greedy iterative refinement algorithm which improved the partition as shown in the numerical results. We also suggested an initial partitioning scheme to improve the performance of the algorithm by avoiding poor local minima. Finally, we identified some ideas to enhance the algorithm which we plan to consider as part of our future work.

REFERENCES

[1] R. Bagrodia, R. Meyer, M. Takai, Yu-An Chen, Z. Xiang, J. Martin, H. Song. PARSEC: a parallel simulation environment for complex systems. In *IEEE Computer*, Vol. 31(10), 1998.

[2] R. Bulo. Game-theoretic framework for similarity-based data clustering. Ph.D. dissertation, Universita CaFoscari Venezia, 2009.

[3] A. Boukerche, S. Das. Dynamic Load Balancing Strategies for Conservative Parallel Simulation. In *Proc. PADS*, pp. 20-28, 1997.

[4] I. Caragiannis, M. Flammini, C. Kaklamanis, P. Kanellopoulos and L. Moscardelli. Tight Bounds for Selfish and Greedy Load Balancing. In *Proc. 33rd International Colloquium on Automata, Languages and Programming*, pp. 311322, 2006.

[5] P. Cramton, Y. Shoham, R. Steinberg. Combinatorial Auctions. *The MIT Press, Cambridge, Massachusetts*, 2006.

[6] M. Garey, D. Johnson. Computers and Intractability. *W. H. Freeman and Company, New York*, 1979.

[7] G. Karypis, V. Kumar. Parallel Multilevel K-way Partitioning for Irregular Graphs. In *SIAM Review*, 41(2):278300, 1999.

[8] G. Karypis, V. Kumar. MeTiS: A software package for partitioning unstructured graphs, partitioning meshes, and computing fill-reducing orderings of sparse matrices. *Univ. of Minnesota, Dept. of Computer Science*, Version 4.0, 1998.

[9] B. Kernighan, S. Lin. An efficient heuristic procedure for partitioning graphs. In *The Bell System Technical Journal*, 49(2):291307, 1970.

[10] A. Kurve, C. Griffin, G. Kesidis. Iterative Partitioning Scheme for Distributed Simulation of Dynamic Networks. *submitted*, January 2011.

[11] J. Lui, M. Chan. An efficient partitioning algorithm for distributed virtual environment systems. In *IEEE Trans. on Parallel and Distributed Systems*, 13(3), March 2002.

[12] B. Nandy, W. Loucks. On a Parallel Partitioning Technique for use with Conservative Parallel Simulation. In *Proc.* 7^{th} Workshop on Parallel and Distributed Simulations, pp. 43-51, 1993.

[13] NetLogo itself: Wilensky, U. 1999. NetLogo. http://ccl.northwestern.edu/netlogo/. Center for Connected Learning and Computer-Based Modeling, Northwestern University. Evanston, IL.

[14] D. Nicol, R. Fujimoto. Parallel Simulation Today. In *Ann. Operations Research*, pp. 249285, 1994.

[15] N. Nisan et al. Algorithmic Game Theory. *Cambridge University Press*, 2007.

[16] H. Ohsaki, G. Oscar, and M. Imase. Quasi-dynamic network model partition method for accelerating parallel network simulation. In *Proc. IEEE MASCOTS*, pages 255-264, 2006.

[17] P. Peschlow, T. Honecker, and P. Martini. A flexible dynamic partitioning algorithm for optimistic distributed simulation. In *Proc. PADS*, pp. 219-228, 2007.

[18] K. Psounis, R. Pan, B. Prabhakar, and D. Wischik. The Scaling Hypothesis: Simplifying the Prediction of Network Performance using Scaleddown Simulations. In *Proc. ACM HOTNETS*, Oct. 2002.

[19] R. Rosenthal. A class of games possessing pure strategy Nash equilibria. In *International Journal of Game Theory*, 2:65-67, 1973.

[20] S. Thulasidasan, S. Kasiviswanathan, S. Eidenbenz and P. Romero. Explicit Spatial Scattering for Load Balancing in Conservatively Synchronized Parallel Discrete-Event Simulations. In *Proc. PADS*, pp. 1-8, 2010.

[21] D. Xu, M. Ammar. Benchmap: benchmark-based, hardware and model-aware partitioning for parallel and distributed network simulation. In *Proc. IEEE MASCOTS*, pp. 455463, 2004.

APPENDIX

The choice of focal nodes is critical to get somewhat equal partitions with high probability. The focal nodes are chosen so that they are at a maximum geodesic distance from each other. More precisely every focal node should be at least $2N_{\frac{|V|}{K}}$ geodesic distance away from all other focal nodes, where $N_{\frac{|V|}{K}}$ is the mean number of hops that cover $\frac{|V|}{K}$ nodes. In this case, we assume that we know the properties of the underlying graph for calculating $N_{\frac{|V|}{K}}$. We would simply like to have the focal nodes as far as possible (in geodesic distance) from each other. This will ensure that the partitions formed after hop-by-hop expansion are somewhat equal. Suppose $F = \{f_1, f_2, ..., f_K\}$ is the set of focal nodes. Hence we would like to have

$$F = \underset{H \subseteq V \ s.t. \ |H|=K}{\arg\max} \ \min_{h,l \in H : l \neq h} d_G(h, l), \tag{5}$$

where d_G is the geodesic distance between the two nodes. We can attempt to find such nodes using heuristics. A simple heuristic could be the one which starts by assigning an arbitrary set of distinct nodes as focal nodes to each processor. Each processor takes a turn sequentially at finding a node from the set of the neighboring nodes that is further away from other focal nodes . This becomes the new focal node of the processor. This process is iterated until we obtain the local maximum of F. We iterate this process over multiple initializations of the focal node set and the maximal set of focal nodes is identified.