Parallel Processing Letters

Superposition and Partitioning for Multi-Source Grid Scheduling with Divisible Loads

Kai Wang, Theron Lammie, Thomas G. Robertazzi, Fellow IEEE
Electrical Engineering Department, Stony Brook University, 100 Nicolls Road Stony Brook, 11790, USA

Taeyoung Lim
KT Future Technology Laboratory, Woomyun 17 Seou, 137-792, Korea

and

Dantong Yu
Computational Science Center, Brookhaven National Laboratory Upton, 11973, USA

Received (received date)
Revised (revised date)
Communicated by (Name of Editor)

ABSTRACT

The applicability of min cost flow and multi-commodity flow mathematical programming problems to steady state, multi-source divisible load scheduling is examined. Applying the linear model concept of superposition to such steady state multi-source load distribution is suggested for linear and more general topologies for the first time. Finally, the use of heuristic optimization for both steady state and transient multi-source load distribution problems is discussed and examples are provided.

Keywords: divisible loads scheduling, grid computing, scheduling, superposition.

1. Introduction

Over the past 20 years [1, 2, 53, 54] a good deal of research has been conducted on scheduling and load distribution with divisible loads. A divisible load is a data parallel load that can be arbitrarily partitioned among links and processors to gain the advantage of parallel processing. However most of this research has involved load distribution from a single source [3, 4]. That is, load originates from a single node in a larger grid or network. Multi-source load scheduling has received less attention but is a logical next step for research in this area. Not only can load be expected to originate from multiple sources in a grid, but even in a supercomputer like the IBM Bluegene load is injected into the fabric of the machine from multiple

points.

An expanding body of work on grid scheduling, has been created during this decade. Some representative work will now be noted. A paper on general architectures is [27]. Prediction involving local grid scheduling [28], queue wait times [29] and variance [30] have been investigated. The decoupling of computation and dats scheduling is examined in [31]. Grid scheduling work has included work on multiple simultaneous requests [32], memory consciousness [33], fault tolerant scheduling [34], incentives [35] and biological concepts [36]. Parameter sweep applications are examined in [37]. Comparisons of grid scheduling algorithms [38] and discussions of the state of the art [39] have appeared. Novel dispatching rules are the subject of [40]. The use of modern heurisitic approaches such as genetic algorithms [41], tabu search and fuzzy logic [42] has received attention. Model driven simulation of grid scheduling strategies is examined in [43]. Finally, market-like computational grids are the subject of [44].

Most of this grid scheduling work involves the scheduling of numbers of atomic indivisible jobs. A *job* is a computational program that can be transported over network links to a remote processor for processing. Typically an indivisible job must be processed on at most one processor. This is different from the divisible load paradigm where load (data) can be arbitrarily partitioned among processors (and links also). This is useful when this is actually the case or as a performance evaluation approximation when the load consists of a large number of small jobs.

A 2002 paper on multi-source load distribution combining Markovian queueing theory and divisible load scheduling theory is Ko and Robertazzi [5]. In 2003 Wong, Yu, Veeravalli, and Robertazzi examined multiple source grid scheduling with capacity constraints [6]. Moges, Yu and Robertazzi considered multiple source scheduling for small size models via linear programming and closed form solutions in 2004 and 2005, respectively [7, 8, 25]. Marchal, Yang, Casanova, and Robert in 2004 studied the use of linear programming to maximize throughput for large grids with multiple loads/sources [9, 51]. Robertazzi and Yu discussed multi-source load distribution in terms of superposition and graph partitioning in 2006 [13]. Li and Veravalli [48] considered multi-source load distribution for single level tree networks in a 2009 journal article. Also, Jia, Veeravalli and Weissman developed algorithms for multi-source load distribution in arbitaray graphs in a 2009 article [49] which used [17] as a starting point.

A distinction is made in this paper between transient and steady state load distribution. *Transient* load distribution involves a finite amount of load and a finite time period during which the distribution and processing occurs. An optimal solution for a transient problem involves doing this distribution and processing in a minimal amount of time. A *steady state* problem involves a continuous flow of load for transport and processing. An optimal solution for a steady state problem often involves maximizing throughput. There is work on both types of problems in the literature [3, 4, 15, 45–47].

Our Contribution

This paper is significant for proposing new optimization approaches for the multiple source, divisible load, scheduling problem in grids. To the authors' knowledge the concept of superposition and the partitioning for multi-source divisible load distribution in general networks is novel. The use of these techniques has the potential to simplify the conceptualization and computation of divisible scheduling problems.

This Paper

This paper proposes the use of min cost flow and multi-commodity flow formulations for steady state divisible load scheduling with multiple sources (section 2). It then goes on in section 3 and 4 to discuss for the first time in the divisible load theory literature the use of superposition techniques, as used in electric circuit theory, for steady state divisible load scheduling in linear and more general topologies, respectively. Also in section 4 a heuristic approach for minimal time solution for both steady state and transient divisible load models is discussed. An example of its use, using genetic optimization, appears in section 5. A second example of superposition is discussed in section 6. The conclusion is in section 7.

2. Flows as Mathematical Programming Problems

A *divisible load* is a data parallel load that can be arbitrarily (with infinitely fine granularity) partitioned among links and processors to gain the advantage of parallel processing. The data in divisible loads have no precedence relationships. Thus the divisible load theory paradigm is data oriented and applied to "embarrassingly parallel" problems. Proposed applications [3,15] for divisible load processing include petascale and exascale experimental data processing, multimedia, wireless, large databases, signature searching [14], DNA processing [26], video processing [19–21], large scale grids [18], sensor nets [17] and image processing [22].

Agrawal and Jagadish [2] were the first to show that optimal load distribution and scheduling for divisible loads can be phrased as mathematical programming problems.

Here optimal load distribution when there are multiple sources of steady state load is formulated two ways: as a minimum cost flow problem and as a more general multi-commodity flow problem. Here a *flow* is a continuous stream of load sent over a link from a source to a sink. A *commodity* is the physical quantity that flows. Here flow and commodity are well known concepts in operations research in the study of mathematical programming. An advantage of using a minimum cost flow problem formulation is that costs can be assigned that are proportional to the amount of flow on each link. Here we assume that there is a single class of flow that is generated at sources and can be processed at any sink.

Let a flow on a link between adjacent nodes i and j be x_{ij} and the "cost" of the flow be c_{ij}. Then let the objective function for the minimum cost flow problem be

$$Z = \sum_{(i,j)\in A} c_{ij}x_{ij} \tag{1}$$

Here A is the set of links and the summation is over all links, also Z is the total

cost.

Next, at each ith node one can write a mass balance constraint [10] that says that the difference between the flow out of node i (overs all links leaving node i) and the flow into node j (overs all links to node j) is, thus

$$\sum_{j:(i,j)\in A} x_{ij} - \sum_{j:(j,i)\in A} x_{ji} = b(i) \; for \; all \; i \in N \tag{2}$$

Here N is the set of nodes. If $b(i)$ is positive the ith node is a source and generates load. If $b(i)$ is negative, the ith node is a sink and processes load. Thus by setting the $b(i)$, the amount of load generated and processed at differences nodes can be include in the optimization. Note that steady state flows are being modeled here. Note also that often $\sum_{i\in N} b(i) = 0$.

With a final set of constraint equations one also optionally set lower (l_{ij}) and upper (u_{ij}) limits to the flow on the ijth link, that is

$$l_{ij} \leq x_{ij} \leq u_{ij} \; for \; all \; (i,j) \in A \tag{3}$$

Solution algorithms, for the minimum cost flow problem are simpler than for multi-commodity flow problems. However in phrasing the optimization as a multi-commodity flow problem one can process specific classes of load at specific processors. Le the kth load class be one of K classes. Then the objective function, which still retains the minimum cost flavor, is

$$Z = min \sum_{(i,j)\in A} \sum_{1\leq k\leq K} c_{ij}^k x_{ij}^k \tag{4}$$

Here c_{ij} and x_{ij} are the ijth link cost and flow, respectively, for the kth class,

A mass balance equation similar to (2) can be written for each Kth flow. Using the notation of Ahuja [10] one has

$$NX^k = b^k, \; k = 1, 2, \cdots, K \tag{5}$$

Thus depending on the sign of the b^k entries, there can be generation and processing of the kth class flow at each ith node. Optionally, upper limits can be placed on the amount of individual class flows and total flow across all classes on a link as the following two sets of constraint equations indicate:

$$0 \leq x_{ij}^k \leq u_{ij}^k \; for \; all \; (i,j) \in A \; and \; all \; k = 1, 2, \cdots, K \tag{6}$$

$$\sum_{1\leq k\leq K} x_{ij}^k \leq u_{ij} \; for \; all \; (i,j) \in A \tag{7}$$

A final note is that an inequality can be included in the mass balance equation if the network/grid has more capacity for processing than the supply of load.

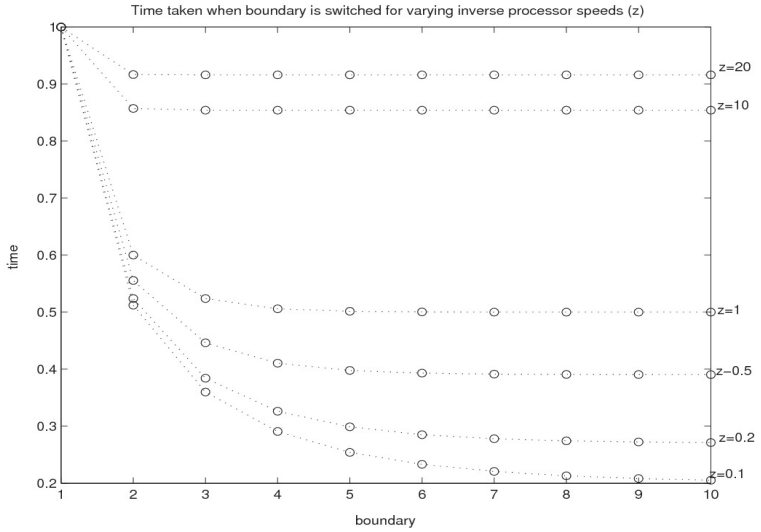

Fig. 1. Finish time (makespan) for linear daisy chain network with equal amounts of divisible load originating at both boundary processors.

3. Linear Daisy Chains with Two Sources

In this and the following sections we examine a specific multi-source problem involving multiple sources with steady state load distribution.

As a tractable motivating example, consider N nodes in a linear daisy chain with divisible load originating at both ends of the chain [11]. For each instance of the problem a boundary node m is established such that load from the left source is processed by nodes 1,2...m and load from the right source is processed by nodes m+1,m+2...N. In effect there are two single source problems (with load originating at a boundary) with the overall solution time given by the maximum of the two single source solution times. The store and forward scheduling strategy of [1] is used for each single source problem.

Figure 1 is a plot of overall network solution time versus the boundary location in a 19 node network for various values of inverse (homogeneous) link speed, z. Each source node supplies an equal amount of load here so that the solution time optimal boundary is at m=10. Note that since the single source solution time versus the number of processors curve is monotonically decreasing, for equal load amounts at each source the overall (dual source) solution time curve is equal to the single source solution time curve for the smaller number of processors.

However this example illustrates a scheduling policy that in most cases is not optimal (the left side and right side do not stop computing at the same time). Can we be more systematic?

It has been noted before that basic divisible load scheduling theory is a linear theory. Other linear theories such as electric circuit theory admit a principle of *su-*

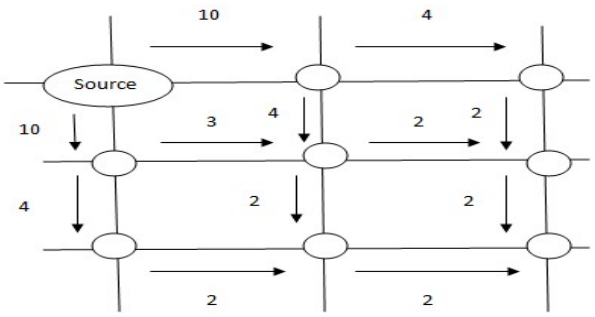

Fig. 2.

perposition. That is, if superposition holds, the response of a network to multiple sources of excitation is equal to the sum of the responses to each source of excitation individually. This is a standard technique used in linear electric circuit theory. Superposition is a property of linear systems in general. A linear system admits superposition and if superposition holds, one has a linear system.

To date the theory of divisible loads has been quite successful in solving load distribution problems with a single source of load. Thus if superposition could be applied in the divisible load scheduling area, it would provide a useful means of computing the flow of load in different parts of a grid when there are multiple sources of load.

Figure 2 and figure 3 are two 3*3 mesh networks. Both of them show one source sending data to other nodes as shown in the figure. With superposition, by adding flows, we can get network flows as in figure 4. To see how this might work in a complete example, consider a linear daisy chain of N processors with load originating in a steady state sense (i.e. load/sec) from the two nodes at either end of the chain.

Consider the chain of Figure 5 with flows from left most and rightmost nodes.

The left most node generates 24 units/second of load while the right most node generates 32 unit/second of load. Hypothetical load distribution patterns for both nodes are shown in the figure. If the two flows are superimposed algebraically the resulting superimposed flow at the bottom of the figure results.

Note that the superimposed flow has a minimum point at the third node from the left. It is straightforward to show that if the two individual flows are monotonically decreasing from source node through the geographic extent of the grid, then the magnitude of the superimposed flow will have a "bath tub" like function shape with a minimum somewhere on the grid. As a degenerate case if the rightward (leftward) flow is greater than the leftward (rightward) flow on all links a monotonically (from left to right) decreasing (increasing) flow magnitude from left to right results.

Let's try now to write some algebraic expressions for the flows in the linear daisy chain with two sources of load on the boundaries. Consider Figure 6.

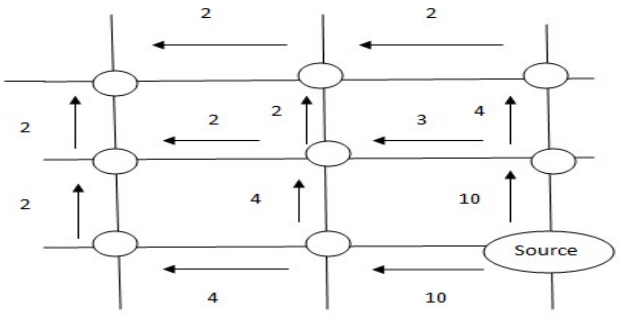

Fig. 3.

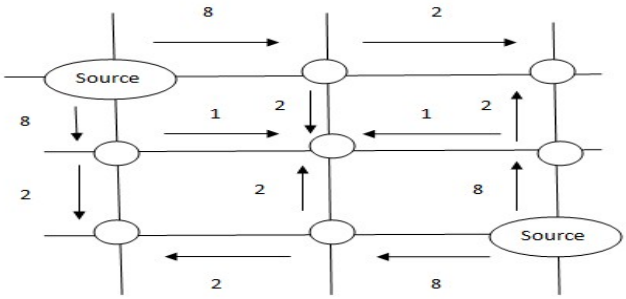

Fig. 4.

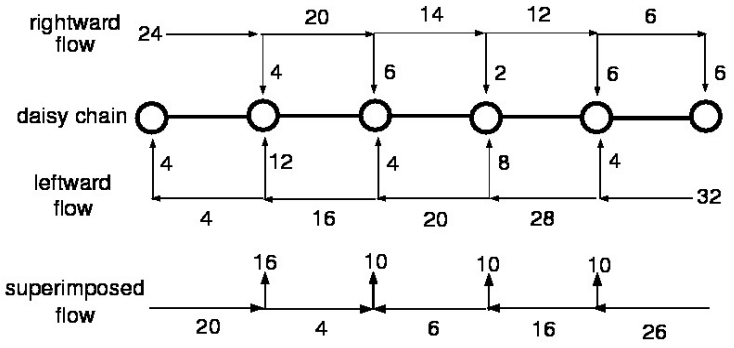

Fig. 5. Superimposed flows in linear daisy chain network.

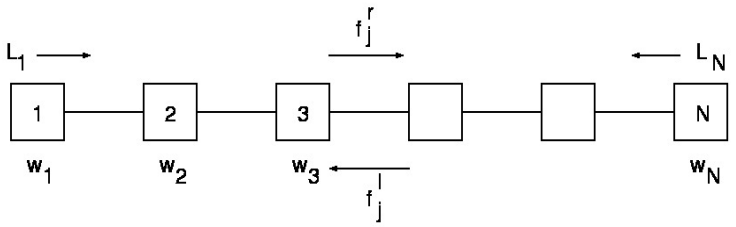

Fig. 6. Linear daisy chain network.

Load in the amount of L_1 units/second originates at the leftmost node while L_N units/second originate at the right most nodes. The rightward flow in the jth link (to the right of the jth node, $j = 1, 2, \cdots, N-1$) is f_j^r. The leftward flow is f_j^l. The inverse available processing speed of the jth node is w_j. We say "available" because in a grid a node may donate only part of its computational capability to process jobs.

We assume for this development that communication time is negligible compared to computation time. Thus it is as if one has links with very large communication speeds. In this case for a solution time optimal distribution of load, load should be distributed to each node in proportion to its computation speed. Over a finite time window load distribution will be balanced so that any other assignment will result in load being over assigned and under assigned to certain nodes and idle times developing. Follow Wong [6], the optimal fraction of load to assign to each node is:

$$\alpha_i^{L_1} = \frac{\frac{1}{w_i}}{\sum_{k=1}^{N} \frac{1}{w_k}} \tag{8}$$

$$\alpha_i^{L_N} = \frac{\frac{1}{w_i}}{\sum_{k=1}^{N} \frac{1}{w_k}} \tag{9}$$

The flow f_j^r at link j is simply the generated load L_1 minus the load consumed by nodes to the left of link j.

$$f_j^r = L_1 - \frac{\sum_{k=1}^{j} \frac{1}{w_k}}{\sum_{k=1}^{N} \frac{1}{w_k}} L_1 \tag{10}$$

$$= L_1 \left(\frac{\sum_{k=j+1}^{N} \frac{1}{w_k}}{\sum_{k=1}^{N} \frac{1}{w_k}}\right) \tag{11}$$

Similarly, for the leftward flow,

$$f_j^l = L_N(\frac{\sum_{k=1}^{j} \frac{1}{w_k}}{\sum_{k=1}^{N} \frac{1}{w_k}}) \tag{12}$$

The magnitude of the superimposed flow is

$$|f_j^r - f_j^l| = \frac{|L_1 \sum_{k=j+1}^{N} \frac{1}{w_k} - L_N \sum_{k=1}^{j} \frac{1}{w_k}|}{\sum_{k=1}^{N} \frac{1}{w_k}} \tag{13}$$

To find the minimum point of the flow magnitude, $|f_j^r - f_j^l| = 0$, so $L_1 \sum_{k=j+1}^{N} \frac{1}{w_k} = L_N \sum_{k=1}^{j} \frac{1}{w_k}$. Therefore,

$$\frac{\sum_{k=1}^{j} \frac{1}{w_k}}{\sum_{k=j+1}^{N} \frac{1}{w_k}} = \frac{L_1}{L_N} \tag{14}$$

Thus the minimum point depends on the ratio of the two generated loads. For instance, if $L_1 = 2L_N$, the minimum point occurs where the amount of load consumed to the left of the minimum point is twice that consumed to the right of the point.

For a continuous and homogeneous version of the problem. (a continuum of nodes on a finite line from 0 to N) one has,

$$\frac{\int_0^x \frac{1}{w} dy}{\int_x^N \frac{1}{w} dy} = \frac{L_1}{L_N} \tag{15}$$

$$L_1(N - x)\frac{1}{w} = L_N x \frac{1}{w} \tag{16}$$

$$x = \frac{L_1}{L_1 + L_N} N \tag{17}$$

This illustrates the same point about the minimum point location, x, just made above.

A natural question is to what extent is the superimposed flow solution optimal. Loads have been allocated to processors in proportion to computation speeds, a time optimal approach. Beyond this the superimposed flows in some sense minimize the amount of load that must be transported between nodes. The superposition technique illustrated here is similar to that an electric circuit with current sources and sinks. A numerical example of superimposed flows appears in section 6.

4. General Networks

The superposition technique of the previous section could be applied to more general grid networks that are two dimensional and that have multiple sources. Consider the grid of Figure 7, the superimposed flow in link ij would be the algebraic sum of the flows due to each of the source individually.

In a very large grid it may make sense not to perform computation for a source at a too distant sink. That is, computation for a source might be done only on nodes

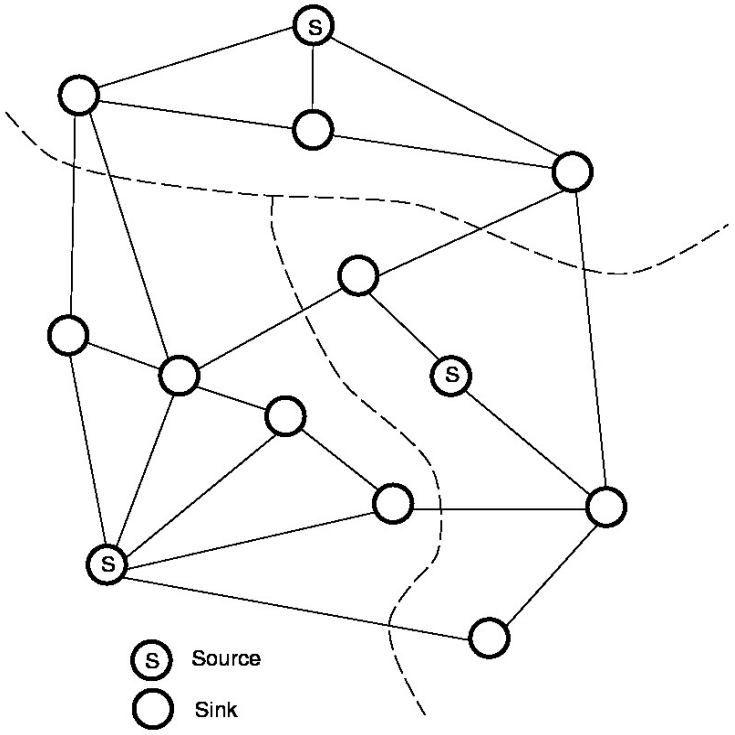

Fig. 7. Partitioned network of processors.

in a local region close to the source. In fact for transient load distribution problems, it has been found for multilevel trees [3], a general spanning network, that for time optimal solutions much more load is processed close to the root (which is the load source) and very little in the further levels of a tree. Here communication delay is taken into account, unlike the situation discussed here.

We close by noting that this suggests an alternate load distribution optimization algorithm to the superposition based approach for both steady-state and transient problems. In this "heuristic" approach the grid is initially partitioned into local regions around each source as shown in Figure 7. The partition is evaluated by running an optimal single source load scheduling algorithm, within each partition. Here optimality is defined in terms of time for a transient problem and throughput for a steady state problem. A new partition could be created by transferring a node from one partition to another according to heuristic rules. Under a greedy strategy, the new partition is kept if it leads to an improved solution. More sophisticated heuristic approaches such as simulated annealing, tabu search or genetic algorithms could also be applied to this transient load distribution optimization problem.

An illustrative example will be provided in the next section that demonstrates some of the modeling/policy possibilities. The policy used is not meant to be optimal

and certainly there are very many possible scheduling policy variations. For length considerations some specific assumptions are made for demonstration purposes. It should be noted that Byrnes and Miller [50] have shown that finding an optimal DLT spanning tree in an arbitrary graph is NP hard. Thus the use of heuristic approaches for general networks is justified.

5. Example: Genetic Algorithm for Partitioning

The implementation of a genetic algorithm in the optimization of a two source node, 100 sink node network example is now discussed (see Figure 5). The parameters are chosen for tractability. Reference [24] contains an earlier, related work. A transient problem is addressed in this section though a steady state problem could also have been considered. This is because the network partitioning and optimization concept discussed in the previous section is useful for both transient and steady state problems. The algorithm features used in this example are meant to be representative - not necessarily optimal - choices. The intention is to demonstrate the basic concept, leaving further elaboration for future work.

In this example a network like that of Figure 8 is assumed. Load distribution is sequential (from each source to each child one after another) and uses staggered start (a child starts processing when all of its load is received). The processor speeds and link speeds are largely randomly chosen in a heterogeneous manner described below.

1. Initial Partitioning Each sink node is added, one at a time, into one of the two partitions while minimizing the total computing speed of all sink nodes in each partition. This procedure can be simply modified to allocate sink computer power to each partition in proportion to amount of load to be distributed from each source node.

2. Fitness Function The following closed form solution for the finish time (i.e. makespan) of each partition with a sequential distribution and a staggered start scheduling policy [3, 52] is used as a fitness function in evaluating each partition. Here w_i is the inverse computing speed of the ith processor and z_i is the inverse computing speed of the ith link. Also T_{cp} is the computing intensity constant (a unit load is processed in the ith processor in time $w_i T_{cp}$). Then T_{cm} is the communication intensity constant (a unit load is transported across the ith link in time $z_i T_{cm}$). Finally, $T_{f,m}$ is the finish time (makespan) with m children processors.

$$T_{f,m} = \frac{w_0 T_{cp}}{1 + k_1 \left[1 + \sum_{i=2}^{m} (\prod_{l=2}^{i} q_l)\right]\Big)} \tag{18}$$

$$k_1 = \frac{w_0}{w_1}, \qquad q_i = \frac{[w_{i-1} T_{cp} + z_{i-1} T_{cm}]}{w_i T_{cp}}, \qquad i = 2, 3, \ldots m$$

We seek to minimize makespan across all partitions. To do this the following mutation and crossover operators are introduced.

3. Mutation and Sequencing Under the sequential load distribution policy implemented in this example, load is transferred sequentially from a partition's

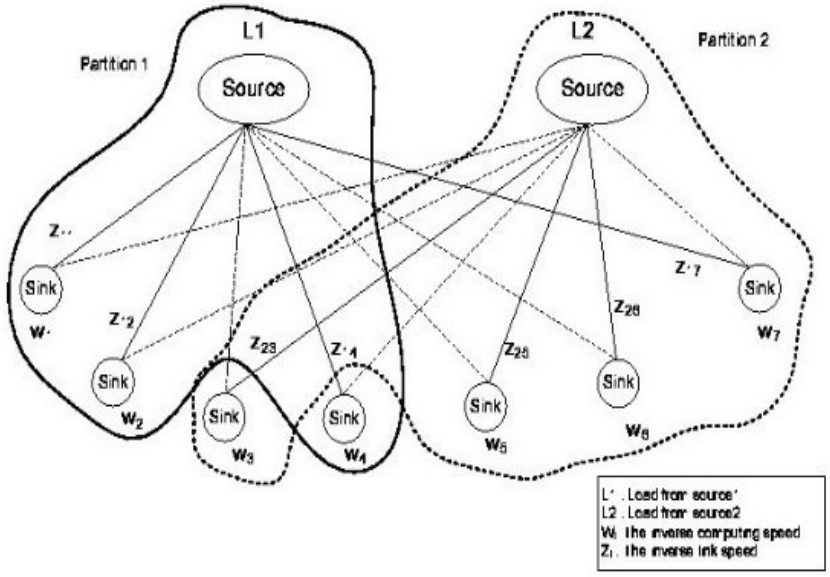

Fig. 8. Two source network.

source to its sinks. The question is then the choice of the best sequence order to minimize makespan for a partition. We simply represent an "individual" in a population of solutions as a vector containing first the source node id for that partition followed by the id's of nodes in sequence distribution order.

Earlier work in 2004 by Charcrannon, Luryi and Robertazzi [12] examined potential heuristic optimization operations such as swapping two adjacent path-processor pairs in a greedy style improvement strategy. In our implementation adjacent sinks and some randomly chosen pairs of sinks are swapped to create new load distribution orders. The number of randomly chosen sinks is equal to the number of sinks in our example optimization. Since all of this results in changes within a given partition, this is a mutation type of operation.

4. Crossover Under a crossover type of operation pairs of sinks are exchanged between partitions to create new individual solutions. A number of such randomly selected exchanges take place each generation and the best (fittest) individual solution resulting from the changes is kept. In our optimization the number of exchanges is set equal to the number of sinks. Note that in situations with heterogeneous processor speeds, swapping nodes with small speeds results in a local improvement and swapping nodes with relatively large speeds results in more potential transformative

impact.

5. Numerical Results We consider two source nodes and a hundred sink nodes. Each sink node is randomly and uniformly assigned and inverse computing speed from 1 to 20 in Figure 9 and is randomly and uniformly assigned an inverse computing speed from 1 to 4 in Figure 10. Each source-sink path is assigned randomly and uniformly an inverse communication speed of from 0.1 to 1. Thus communication is always faster than computing, making distributed computation efficacious.

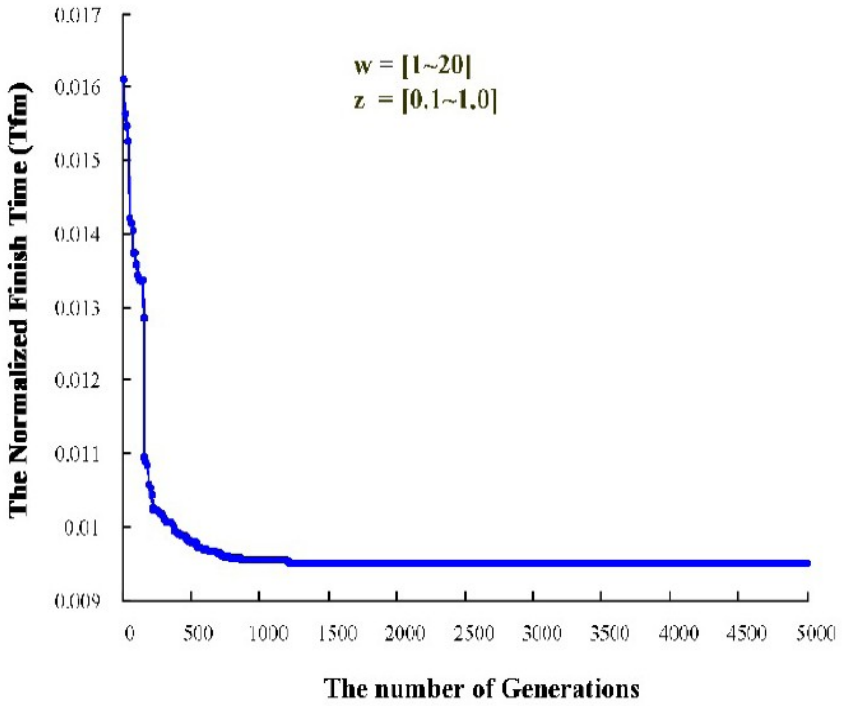

Fig. 9. Finish time versus the number of generations where w=[1,20] and z=[0.1,1.0] (uniform distribtions).

Figures 9 and 10 shows the finish time (makepsan) versus the number of generations for the parameters above. The figures show that the more extreme the difference between communication and computing speeds, the faster a solution results. Figure 11 shows the difference in makepan between the two partitions being driven to zero over the generations.

Running the program in 2007 took 1 minute on a PC. Using Amdahl's law the 2013 running time should be about 4 seconds (100 percent improvement over 18 months). In reality running time is even less as the solution converges in much less than the full time windows shown in Figures 9 and 10.

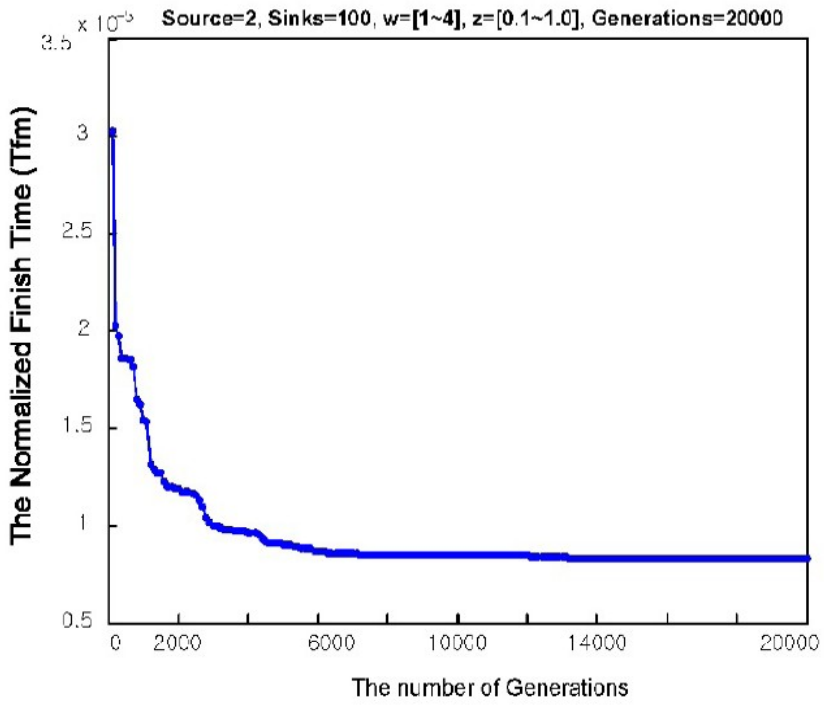

Fig. 10. Finish time versus the number of generations where w=[1,4] and z=[0.1,1.0] (uniform distributions).

Beyond this demonstration example, the development of a very fast optimization algorithm for very large grids, perhaps using a technique different from genetic algorithms, is an interesting problem for future research.

6. Example: Superposition

Now we discuss a simulation example of superposition. First, we create networks and in each network there are N nodes whose coodinates are distributed as Poisson random number between 0 to 100 in a 100*100 square. Also S of those nodes are the source nodes and they send data with an amount of data A. We connect these nodes through a spanning tree. To determine each node's flow, we start from the source as root, the root keeps 1/n of the flow and sends the rest of the load to the its n-1 connected branches. For each source, there is a flow matrix to record the flow on the connections. Then for each connection, we add up the flows in the same direction and cancel the ones in the opposite directions and get the superinposed flow matrix. Then from those networks' flow matrix, we can see the performance of superposition.

Here we have T=300, N=100, A=100, S=30. That is there are T=300 networks,

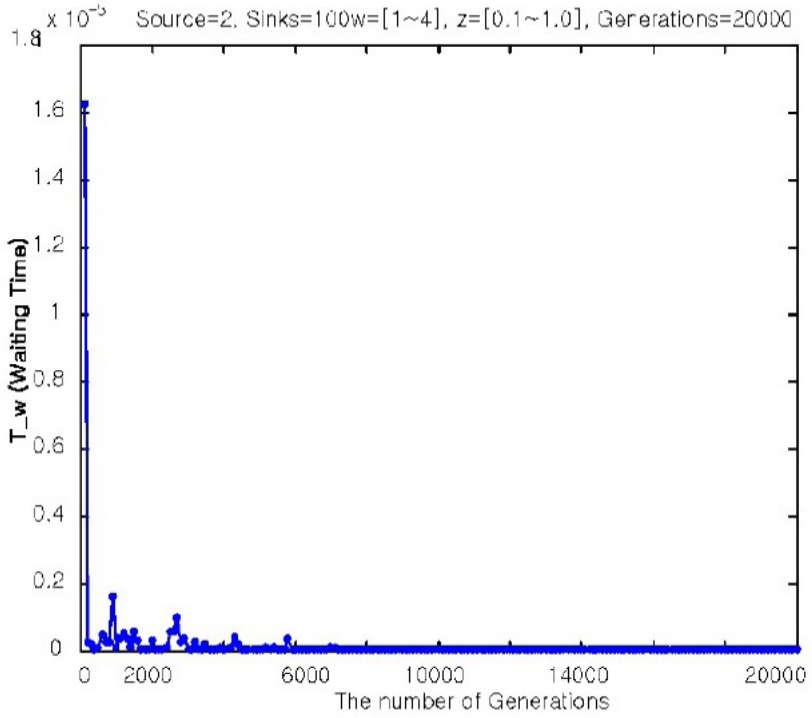

Fig. 11. Difference in makespan between the two partitions versus the number of generations where w=[1,4] and z=[0.1,1.0] (uniform distributions).

in each network there are 100 nodes and every sources' amount of data is 100. Figure 12 to figure 19 are the graphs of performance. In figure 12, the horizotal axis is the number of flows on a link and the vertical axis indicates superimposed flows on the same link over the sum of their absolute value: $\sum_i \text{flow(i)} / \sum_i \left|\text{flow(i)}\right|$. This figure shows that the superposition effect becomes more and more notable as the number of flow increases.

In figure 13, the horizotal axis is the number of flows on a link and the vertical axis is the number of times they appeared. This figure illustrates that the links with smaller number of flows appeared more frequently while the links with larger number of flows have a smaller chance to appear.

Figure 14 shows the trend of the sum of absolute value of flow ($\sum_i \left|\text{flow(i)}\right|$) on the links as the links' distance to the center (x axis) increases. Figure 17 shows the sum of the canceled flow ($\sum_i \text{flow(i)}$) on the links as the links' distance to the center (x axis) increases.

In figure 15 and 18, the horizontal axis is the links' distance to the center of the network and the vertical axis is the flow density which is arrived at by dividing

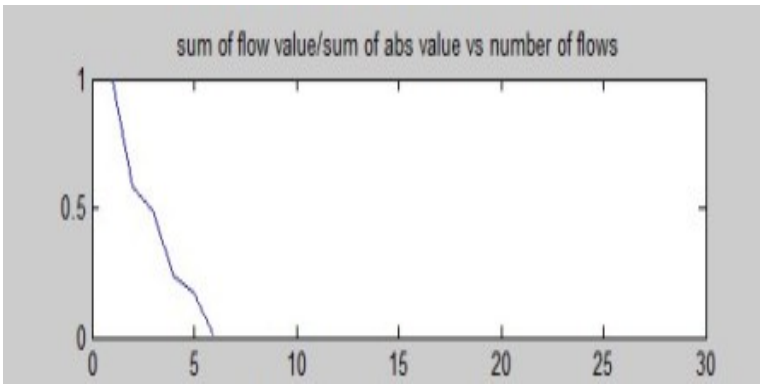

Fig. 12.

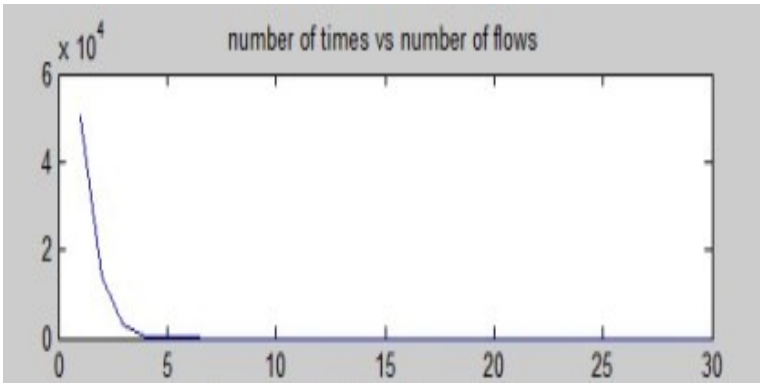

Fig. 13.

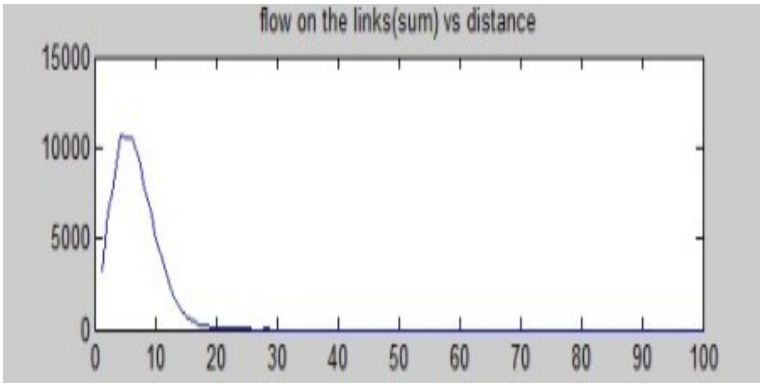

Fig. 14.

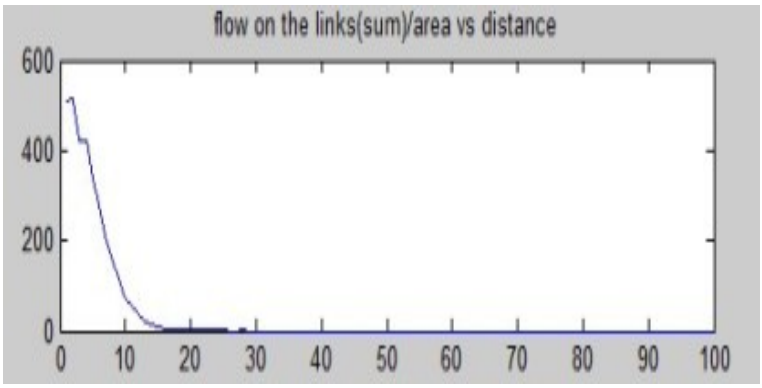

Fig. 15.

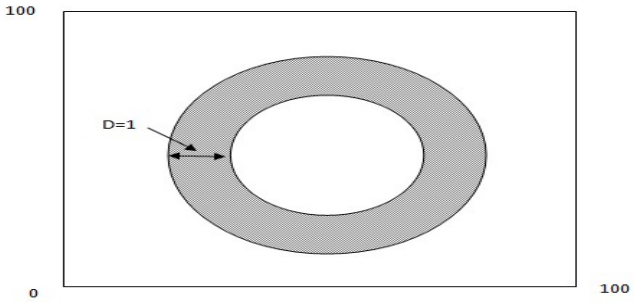

Fig. 16.

the sum of the flows a distance from the center by its area as shown in figure 16. This division by area compensates for the fact that there are more nodes further from the center than closer to the center. Figure 15 is the sum of absolute value of flow divided by the area and figure 18 is the sum of canceled flow divided by the area. This illustrates that the flow density has the highest value in the center of the network and decreases as the distance to the center increases. Node degree verse distance to the center of the network is illustrated in figure 19.

7. Conclusion

Several optimization techniques of the multiple source grid scheduling problem have been outlined. One new technique proposed here, the application of superposition to such problems, is particularly exciting as it brings an established linear modeling technique to a new area. In the process it provides conceptual simplification and the possibility of efficient computation. A second technique, the proposed partition heuristic for general networks, is a compromise between a theoretically optimal

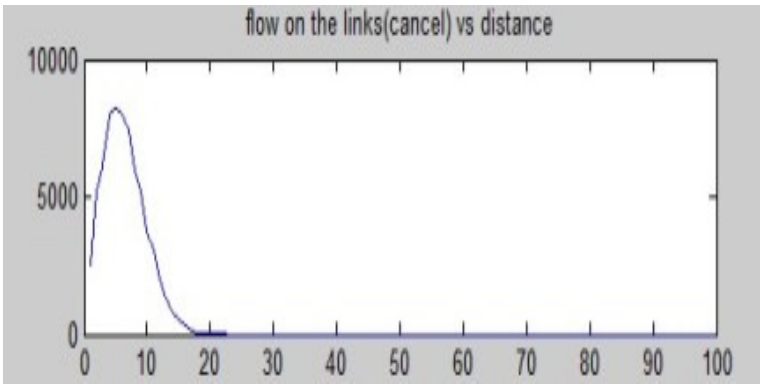

Fig. 17.

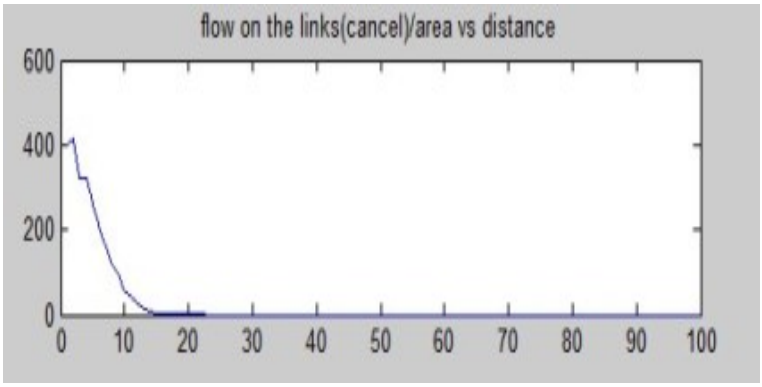

Fig. 18.

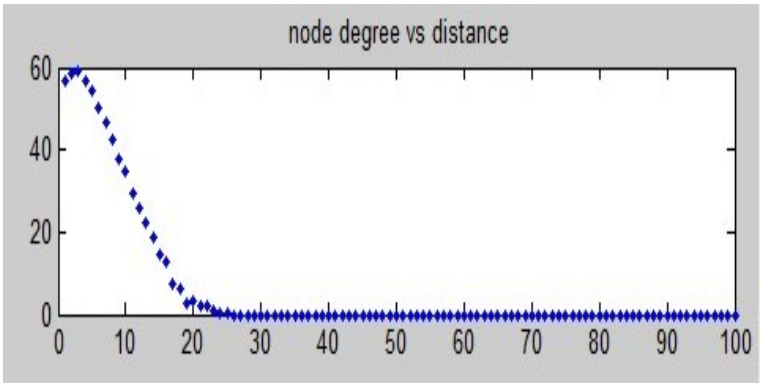

Fig. 19.

superposition solution and practical network behavior. These two techniques should be of interest because of their theoretical, computational and applied utility. Future work includes using the concept of superposition to simplify divisible load scheduling algorithms and the development of very fast algorithms for partitioning very large grids.

8. Acknowledgments

This work was performed while Taeyoung Lim was a Ph.D student at Stony Brook University. The authors thank Professor Ester Arkin of Stony Brook's Applied Mathematics Department for very useful discussions. Dantong Yu and Thomas Robertazzi's work is supported by DOE grant. DE-SC0003361.

References

[1] Y.-C. Cheng and T.G. Robertazzi, "Distributed Computation with Communication Delays," *IEEE Trans. Aerospace and Electronic Systems*, vol. 24, pp. 700-712, Nov. 1988.

[2] R. Agrawal and H.V. Jagadish, "Partitioning Techniques for Large-Grained Parallelism," *IEEE Trans. Computers*, vol. 37, pp. 1627-1634, Dec. 1988.

[3] B. Veeravalli, D. Ghose, V. Mani and T.G. Robertazzi, *Scheduling Divisible Loads in Parallel and Distributed Systems*, IEEE Computer Society Press (now distributed by Wiley), Los Alamitos CA, 1996.

[4] B. Veeravalli, D. Ghose and T.G. Robertazzi, "Divisible Load Theory: A New Paradigm for Load Scheduling in Distributed Systems," in special issue of *Cluster Computing* on divisible load scheduling (D. Ghose and T.G. Robertazzi, eds.), Kluwer Academic Publishers, vol. 6, pp. 7-18, Jan. 2003.

[5] K. Ko and T.G. Robertazzi, "Scheduling in an Environment of Multiple Job Submissions," *Proc. 2002 Conference on Information Sciences and Systems*, Princeton University, Princeton NJ, March 2002.

[6] H. Wong, D. Yu, B. Veeravalli and T.G. Robertazzi, "Data Intensive Grid Scheduling: Multiple Sources with Capacity Constraints," *Proc. IASTED International Conference on Parallel and Distributed Computing and Systems* (PDCS 2003), Marina del Rey, CA, Nov. 2003.

[7] M. Moges, D. Yu and T.G. Robertazzi, "Grid Scheduling Divisible Loads from Multiple Sources via Linear Programming," *Proc. IASTED International Conference on Parallel and Distributed Computing and Systems* (PDCS 2004), Cambridge, Mass., 2004.

[8] M. Moges, D. Yu and T.G. Robertazzi, "Divisible Load Scheduling with Multiple Sources: Closed Form Solutions," *Proc. 2005 Conference on Information Sciences and Systems*, The Johns Hopkins University, Baltimore MD, March 2005.

[9] L. Marchal, Y. Yang, H. Casanova and Y. Robert, "A Realistic Network/Application Model for Scheduling Divisible Loads on Large-Scale Platforms," in *Proc. International Parallel and Distributed Processing Symposium* (IPDPS 2005), IEEE Computer Society, April 2005.

[10] R. Ahuja, T. Magnanti and J. Orlin, eds., *Network Flows*, Prentice-Hall, 1993.

[11] T. Lammie and T.G. Robertazzi, "A Linear Daisy Chain with Two Divisible Load Sources," *Proc. Conference on Information Sciences and Systems*, The Johns Hopkins University, Baltimore MD, March 2005.

[12] S. Charcranoon, S. Luryi and T.G. Robertazzi, "Load Sequencing for a Parallel Processing Utility," *Journal of Parallel and Distributed Computing*, vol. 64, pp. 29-35, 2004 (Research Note).

[13] D. Yu and T.G. Robertazzi, "Multi-Source Grid Scheduling for Divisible Loads," *Proc. Conference on Information Sciences and Systems*, Princeton University, Princeton NJ, March 2006, pp-188-191.

[14] K.Ko and T.G. Robertazzi, "Signature Search Time Evaluation in Flat File Databases," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 44, no. 2, pp. 493-502, April 2008.

[15] T.G. Robertazzi, "Ten Reasons to Use Divisible Load Theory," *Computer*, vol. 36, no. 5, pp. 63-68, Mat 2003.

[16] V. Bharadwaj and N. Viswanadham, "Sub-Optimal Solutions using Integer Approximation Techniques for Scheduling Divisible Loads," *IEEE Transactions on Systems, Man and Cybernetics, Part A*, vol. 30, no. 6, pp. 680-691, Nov. 2000.

[17] M. Moges and T.G. Robertazzi, "Wireless Sensor Networks: Scheduling for Measurement and Data Reporting," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 42, no. 1, pp. 327-340, Jan. 2006.

[18] Viswanathan, B. Veeravalli and T.G. Robertazzi," "Resource Aware Distributed Scheduling Strategies for Largs-Scale Cluster/Grid Systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 18, no. 10, pp. 1450-1461, Oct. 2007.

[19] P. Li, B. Veeravalli and A.A. Kassim, "Design and Implementation of Parallel Video Encloding Strategie using Divisible Load Analysis," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 15, no. 9, pp. 1098-1112, Sept. 2005.

[20] S. Suresh, V. Mani, S.N. Omkar and H.J. Kim, "Parallel Video Processing using Divisible Load Scheduling Paradigm," *Korean Journal of Broadcast Engineering*, vol. 10, pp. 83-102, 2005.

[21] V. Bharadwaj, X. Li and C.C. Ko, "Efficient Partitioning and Scheduling of Computer Vision and Image Processing Data on Bus Networks using Divisible Load Analysis," *Image and Vision Computing*, Elsevier Publishers, vol. 18, no. 11, pg. 919-938, Aug. 2000.

[22] V. Bharadwaj and S. Ranganath, "Theoretical and Experimental Study of Large Size Image Processing Applications using using Divisible Load Paradigm on Distributed Bus Networks," *Image and Vsiion Computing*, Elsevier Publishers, vol. 13 no. 13-14, pp. 917-936, Dec. 2002.

[23] L. Marchal, Y. Yang, H. Casanova and Y. Robert, "A Realistic Network/Application Model for Scheduling Divisible Loads on Large-Scale Platforms," *Proc. International Parallel and Dsitributed Processing Symposium*, 2005.

[24] S. Kim and J.B. Weissman, "A Genetic Algorithm Based Approach for Scheduling Decomposable Data Grid Applications," *Proc. 2004 International Conference on Parallel Processing*, Aug. 2004.

[25] M. Moges, D. Yu and T.G. Robertazzi, "Grid scheduling divisible loads from two sources," *Computers and Mathematics with Applications*, vol. 58, pp. 1081-1092, 2009.

[26] H.M. Wong and B. Veeravalli, "Aligning Biological Sequences on Distributed Bus Networks: A Divisible Load Scheduling Approach," *IEEE Transactions on Information Technology in BioMedicine*, vol. 9, no. 4, pp. 489-501, Dec.

2005.

[27] J. Schopf, "A General Architecture for Scheduling on the Grid," *Argonne National Laboratory preprint* ANL/MCS, Argonne IL 1000-1002, 2002.

[28] D, Spooner et. al., "Local Grid Scheduling Techniques using Performance Prediction," *IEEE Proceedings*, vol. 150, pp. 87-96, 2003.

[29] W. Smith, V. Taylor and I. Foster, "Using Run-time Prediction to Estimate Queue Wait Times and Improve Schedule Performance," *Proceedings of the Job Scheduling Strategies for Parallel Processing Conference* in Lecture Notes in Computer Science, vol. 1659, 1999, pp. 202-219.

[30] L. Yang, J. Schopf and I. Foster, "Conservative Scheduling: Using Predicted Variance to Improve Scheduling Decisions in Dynamic Environments," *Proceedings of Supercomputing 03*, pp. 558-664, 2003.

[31] K. Ranganathan and I. Foster, "Decoupling Computation and Data Scheduling in Distributed Data-intensive Applications," *Proceedings of the 11th Annual IEEE International Symposium on High Performance Distributed Computing*, pg. 352, 2002.

[32] V. Subramani, R. Kettimuthu, S. Srinivasan and P. Sadayappan, "Distributed Job Scheduling on Computational Grids using Multiple Simultaneous Requests," *Proceedinsg of gthe 11th Annual IEEE International Symposium on High Performance Distributed Computing*, pp. 359-366, 2002.

[33] M. Wu and X. Sun, "Memory Conscious Task Partition and Scheduling in Grid Environments," *Proceedings of the Fifth IEEE/ACM International Workshop on Grid Computing*, 2004, pp. 138-145, 2004.

[34] J. Abawajy, "Fault Tolerant Scheduling Policy for Grid Computer Systems," *Proceedings of the 18th International Parallel and Distributed Processing Symposium*, pp. 238-245, 2004.

[35] L. Xiao, Y. Zhu, L. Ni and Z. Xu, "An Incentive-based Grid Scheduling," *Proceedings of the 19th International Parallel and Distributed Processing Symposium*, pg. 65b, 2005.

[36] A. Chakravarti, G. Baumgartner and M. Lauria, "Application-specific Scheduling for the Organic Grid," *Proceedings of the Fifth IEEE/ACM International Workshop on Grid Computing*, pp. 146-155, 2004.

[37] E. Huedo, R. Montero and I. Llorente, "Experience on Adaptive Grid Scheduling of Parameter Sweep Applications," *Proceedings of the 12th Euromicro Conference on Parallel, Distributed and Network-based Processing*, pp. 28-33, 2004.

[38] N. Fujimoto and K. Hagihara, "A Comparison Among Grid Scheduling Algorithms for Independent Coarse-Grained Tasks," *Proceedings of the 2004 International Symposium on Applications and the Internet*, pp. 674-680, 2004.

[39] Y. Jian and Y. Liu, "The State of the Art in Grid Scheduling Systems," *Proceedings of the Third International Conference on Natural Computation*, pp. 619-623, 2007.

[40] A. Rasooli, M. Mirza-Aghatabar and S. Khorsandi, "Introduction of Novel Dispatching Rules for Grid Scheduling Algorithms," *Proceedings of the International Conference on Computer and Communication Engineering*, pp. 1072-1078, 2008.

[41] W. Li and C. Yuan, "Research on Grid Scheduling based on Modified Genetic Algorithm," *Proceedings of the Third International Conference on Pervasive Computing and Applications*, vol. 2, pp. 633-638, 2008.

[42] C. Fayad, J.M. Garibaldi and D. Ouelhadj, "Fuzzy Grid Scheduling using Tabu Search," *Proceedings of the IEEE International Fuzzy Systems Conference*, pp.

1-6, 2007.

[43] H. Li and R. Buyya, "Model-driven Simulation of Grid Scheduling Strategies," *Proceeding of the Third International Conference on e-Science and Grid Computing*, pp. 287-294, pp. 2007.

[44] L. Xiao, Y. Zhu, L. Ni and Z. Xu, "Incentive-Based Scheduling for Market-Like Computational Grids," *IEEE Transactions on Parallel and Distributed Systems*, vol. 19, no. 7, pp. 903-913, July 2008.

[45] O. Beaumont, A. Legrand, L. Marchal and Y. Robert, "Steady-state Scheduling on Heterogeneous Clusters," *International Journal of Foundations of Computer Science*, vol. 16, no. 2, pp. 163-194, 2005.

[46] D. Bertsimas and D. Gamarnik, "Asymptotically Optimal Algorithm for Job Shop Scheduling and Packet Routing," *Journal of Algorithms*, vol. 33, no. 2, pp. 296-318, 1999.

[47] A. Legrand, L. Marchal and Y. Robert, "Optimizing the Steady-State Throughput of Scatter and Reduce Operations on Heterogeneous Platforms," *Journal of Parallel and Distributed Computing*, vol. 65, pp. 1497-1514, 2005.

[48] X. Li and B. Veeravalli, "PPDD: Scheduling Multi-Site Divisible Loads in Single-Level Tree Networks," *Vol 13, in Cluster Computing*, 2010, pp-31-96.

[49] J. Jia, B. Veeravalli and J. Weissman, "Scheduling Multi-source Divisible Loads on Arbitrary Networks," *Vol. 21, no.4 in IEEE Transactions on Parallel and Distributed Systems*, 2010, pp-520-531.

[50] P. Byrnes and L.A. Miller, "Divisible Load Scheduling in Distributed Computing Environments: Complexity and Algorithms," *Technical Report MIN-ISYE-06-006*, University of Minnesota Graduate Program in Industrial and Systems Engineering, 2006.

[51] L. Marchal, Y. Yang, H. Casanova and Y. Robert, "Steady-state Scheduling of Multiple Divisible Load Applications on Wide-Area Distributed Computing Platforms," *International Journal of High Performance Computing Applications*, vol. 20, no. 3, pp. 365-381, 2006.

[52] T. Robertazzi, *Networks and Grids: Technology and Theory*, Springer, 2007.

[53] M. Drozdowski *Scheduling for Parallel Processing*, Springer, 2009.

[54] H. Casanova, A. Legrand and Y. Robert. *Parallel Algorithms*, CRC Press, 2009.