

# **Resource-Aware Load Balancing of Parallel Applications**

Eric Aubanel  
Faculty of Computer Science  
University of New Brunswick  
Fredericton, NB  
Canada  
voice: +1 506-458-7268  
fax: +1 506-453-3566  
email: [aubanel@unb.ca](mailto:aubanel@unb.ca)

*Revised version accepted February 2008 by the Encyclopedia of Grid Computing*

# Resource-Aware Load Balancing of Parallel Applications

## INTRODUCTION

Distributed high performance computing (HPC) applications have formed an important class of grid applications from the early days of the I-Way (DeFanti et al., 1996, Foster & Kesselman, 1999) to the TeraGrid (<http://www.teragrid.org>) of today. The main reason is that the aggregation of multiple parallel computers permits problem solutions that require more resources than are available in a single system. Many of these applications, such as partial differential equation (PDE) solvers, can be described by an undirected graph representing concurrent execution of tasks and communication between tasks, as in Foster's PCAM design methodology (Foster, 1995). Parallel execution requires partitioning of the application's graph in such a way that communication between the resulting subgraphs is minimized and the load is roughly balanced. These subgraphs must be mapped to the processors of a parallel computer. In many cases partitioning is only required once before execution of the application begins. However, in situations where the computational requirement of tasks varies with time, as in adaptive mesh refinement methods, the graph may need to be repartitioned to re-balance the computational load of processors (Schloegel, Karypis, & Kumar, 2003; Teresco, Devine, & Flaherty, 2005). Here we use the term "load balancing" to refer to both static partitioning and dynamic repartitioning. This load balancing problem becomes particularly challenging when a homogeneous parallel computer is replaced by a heterogeneous computational grid.

Load balancing using graph partitioning is a well-established research area that has resulted in a number of popular software tools used by computational scientists (Schloegel, Karypis, & Kumar, 2003). Several approaches have been proposed for this NP-hard problem. Many employ a multilevel approach (Schloegel, Karypis, & Kumar, 2003), which collapses (coarsens) the graph recursively, partitions the smallest graph, and refines the partition as the graph is uncoarsened. One critique of classical graph partitioners is that minimizing the number of edges cut by a partition misrepresents the actual communication volume (Hendrickson, 1998), but in many cases this metric still provides a reasonable estimate of the total communication volume of the parallel application. In reality, however, it is the time (due to computation and communication) spent by the slowest processor that determines the execution time of a parallel application. This becomes particularly important when the computational platform is heterogeneous in processor and network performance, in which case the number of edges cut by a partition is a poor measure of parallel overhead. We call resource-aware load balancing the partitioning and mapping of an application's graph to a heterogeneous computational platform, either before execution begins or periodically during execution.

Recent work has begun to address the problem of resource-aware load balancing. This includes DRUM, JOSTLE, MinEX, PaGrid, PART, and SCOTCH, among others. Our goal is to review and contrast these recent efforts and discuss the many avenues for future work. A comprehensive review of this area has not been published to date (Li & Lan, 2004; Devine et al., 2005).

## BACKGROUND

Since heterogeneous computing is a broad research area and there are many definitions of computational grids, we define the characteristics of the applications and platforms that we are considering.

### Applications

We consider tightly-coupled parallel applications that can be described by an undirected graph representing concurrent execution of tasks and communication between tasks. The most important class concerns the numerical solution of PDEs (Schloegel, Karypis, & Kumar, 2003), but such applications also include molecular dynamics problems (Koenig & Kalé, 2007) and cellular automata (Cappuccio et al., 2001). Parallel execution requires partitioning of the application's graph in such a way that communication between the resulting subgraphs is minimized and the load is roughly balanced. These applications are typically iterative, with alternating communication and computation phases (Botadra et al., 2007). Although partial overlap is possible, the entire communication overhead cannot be overlapped with computation because of the dependency between subsequent iterations. Scheduling of operations on dense matrices on heterogeneous systems has been studied (Dongarra & Lastovetsky, 2006), but is not considered here. We are also not concerned here with scheduling of independent tasks or scheduling of workflows represented by directed graphs, but with scheduling of the components of a single parallel program, defined as the use of multiple resources executing concurrently to solve a single problem.

### Platforms

*High capacity* grid platforms are dedicated to high performance computing (HPC) applications and generally consist of an aggregation of multiple HPC clusters (e.g. the TeraGrid) and are space-shared, that is processors only execute a single user's process(es) at a time. The clusters may be located in one organization, or they may be more spread out geographically, which can pose additional problems of high latencies. *High throughput* grid platforms aim to take advantage of spare compute cycles on processing nodes that may be dedicated to multiple users at the same time and hence are time-shared. This latter class can consist of local or global networks of computers, and can be either within one organization (Chien, 2003) or be simply a collection of individual computers on the Internet (Anderson, 2004). While high throughput grids are mainly useful for scheduling independent tasks they can also be used for parallel applications, and therefore both types of grids will be considered here. For more on classification of computational grids, see Dongarra & Lastovetsky (2006) and Stockinger (2007).

Grid computing platforms can be heterogeneous in many ways, whether due to varying processor and network link performance, available memory, or system software, to name a few (Dongarra & Lastovetsky, 2006). Here we consider only processor and network heterogeneities, since they alone significantly complicate the load balancing problem. Such heterogeneities can be static or dynamic, and the load balancing strategies differ significantly for both cases. Processor heterogeneity can be static due to the presence of processors of varying potential performance, and can be dynamic for time-shared platforms. Network heterogeneity can be static due to the presence of slow and

fast links, and can be dynamic due to the varying performance over time of wide-area networks. It should be noted that the load balancing algorithms discussed here are not only applicable to grids, but are also valid for single clusters that have static or dynamic heterogeneities.

## MAIN FOCUS

Since the load balancing problem is NP-hard, exact solutions are only possible for graphs too small to be useful (see, e.g., Attiya & Hamam, 2004, and references therein), since a typical application graph will have well over one thousand vertices. Load balancing algorithms fall into two classes: static and dynamic. Resource-aware algorithms also consider static and dynamic processor and network characteristics.

### Static Load Balancing

The simplest type of heterogeneity to deal with is due to fixed differences between processors. Varying processor performance can be accommodated by weighting the size of subgraphs of a partition, which can be easily incorporated into homogeneous partitioners, such as METIS (Karypis & Kumar, 1998). Accommodating static network heterogeneity is more challenging, and a number of approaches have been taken. These can be classified into two classes: partitioning and mapping, and hierarchical partitioning.

A number of partitioners, including SCOTCH, JOSTLE, and PaGrid, partition and map the application graph onto a weighted platform graph. To our knowledge, SCOTCH (Pellegrini & Roman, 1996) was the first partitioner of this type. Mapping of graphs to certain types of platform graphs (e.g. hypercube) has been well studied (Grama et al., 2003), but here we are considering mapping of arbitrary application graphs to arbitrary platform graphs. SCOTCH employs recursive bisection of both the mesh and platform graphs. At each step the two graphs are bipartitioned, while maintaining a balanced load of vertices, and a subset of vertices is mapped onto a subset of processors when the size of the subset of processors is equal to one. JOSTLE (Walshaw & Cross, 2001) is a multilevel graph partitioner that uses a variant of the Kernighan-Lin algorithm and a cost function based on total communication cost (as does SCOTCH). While JOSTLE produces partitions that are balanced in terms of vertices of the application graph, it does not take into account the communication cost of each processor when load balancing. This can produce imbalanced partitions, with processors connected by slow links having higher execution times than the rest (Wanschoor & Aubanel, 2004). The initial version of PaGrid (Huang, Aubanel & Bhavsar, 2003, 2006) used a multilevel graph partitioning approach, with refinement based on minimization of total communication cost, as in JOSTLE, augmented by a load balancing stage based on estimated execution time in the final uncoarsening phase. The next version (Wanschoor & Aubanel, 2004) improved PaGrid by using the estimated execution time cost function at all levels of refinement. PaGrid was further improved by incorporating latency into the estimated execution time (Aubanel & Wu, 2007). Adams & Price (2004) proposed a Boltzmann machine neural network for the partitioning and mapping problem, in which multiple simulated annealings are performed. Their approach lends itself to parallelization, but may still be costly for large problems. Recent work by Moulitsas & Karypis (2006) has investigated adapting the METIS algorithm to first balance the load of processors, followed by an additional stage that modifies this partition to minimize the communication volume. They also discuss briefly the possibility of using a similar approach to PaGrid, where the estimated execution times of processors is balanced and the maximum time is minimized.

The hierarchical partitioning approach has been taken by projects such as DRUM, Charm++, PART, and by the work of Otero et al. (2005). The first two are dynamic load balancers and will be discussed in the next section. PART (Chen & Taylor, 2002) uses simulated annealing, with a cost function based on estimated execution time. It first partitions the mesh into  $c$  subgraphs, where  $c$  is the number of clusters, then partitions each subgraph for the processors in each cluster, and finally performs a global retrofit. This computationally intensive algorithm requires a parallel implementation. Otero et al. (2005) use METIS to partition the graph into  $c$  subgraphs, and then partition each subgraph into two classes: subgraphs containing only boundary nodes communicating with other clusters and those containing only communications local to the cluster. This allows slow remote communications to be overlapped with computation.

While many problems involve dynamic applications and/or dynamic platforms, and hence require dynamic load balancing, static load balancing is still important, since initial assignment of tasks is crucial to the performance of dynamic load balancing. As well, many algorithmic techniques developed for static partitioners can also be applied to dynamic ones.

### **Dynamic Load Balancing**

These algorithms have traditionally been designed to handle parallel applications where during execution either the mesh is adaptively refined (AMR) or the computational load varies over time. They begin with a static partitioning of the initial graph, then periodically monitor the load of processors, followed by reassignment of vertices to balance the load as required. This reassignment can be accomplished either by graph repartitioning or by diffusion algorithms (Schloegel, Karypis, & Kumar, 2003; Teresco, Devine, & Flaherty, 2005). These algorithms must not only balance the load and minimize inter-processor communication in the parallel application, but also must minimize the cost of reassigning vertices, which can be significant.

Dynamic load balancing algorithms can easily be extended to deal with dynamic processor heterogeneity. Minyard & Kallinderis (2000) applied an octree-based method for the initial partition and repartitions of a computational fluid dynamics application. They used runtime measurements to determine the need for load balancing due either to mesh adaptation or to changes in the runtime environment. Rao (2006) modified an explicit finite element solver to periodically check the load of processors and then used a simple load balancing algorithm. He indicated that a more scalable solution could use diffusion algorithms to minimize redistribution cost. Dobber, Koole, & van der Mei (2004) performed experiments on dynamic load balancing of a red/black SOR algorithm on the Planetlab computational grid (<http://www.planet-lab.org>), using exponential smoothing to predict processor performance.

Network heterogeneity can be taken care of directly, in a manner similar to the graph partitioning and mapping algorithms discussed above, or indirectly, by performing hierarchical partitioning. In the case of load balancers that employ diffusion algorithms, care should be taken to include the varying performance of network links when evaluating redistribution cost (Rao, 2006; Rotaru & Nageli, 2004). The GrACE adaptive runtime system supports AMR applications on dynamic heterogeneous systems (Sinha & Parashar, 2002). Load balancing is done using a capacity metric determined for every processor, from the system state monitored using the Network Weather Service tool (<http://nws.cs.ucsb.edu>). MinEX (Das, Harvey, & Biswas, 2002) is a multilevel dynamic partitioner that minimizes the estimated execution time of the application. It refines partitions by moving vertices from overloaded to underloaded processors in order to

minimize the variance in processor execution times. Petcu, Vizman, & Paprzycki (2006) used a heuristic partitioning algorithm called Largest Task First with Minimum Finish Time and Available Communication Cost for load balancing of a multi-block CFD application in a heterogeneous computing environment. Here grid blocks are scheduled to processors, taking into account dynamically varying performance of the platform.

Hierarchical approaches are commonly used in this area. The Dynamic Resource Utilization Model (DRUM) is used in Zoltan, a parallel computing toolkit, to collect information about the processors and network of the computing platform (Devine et al., 2005; Teresco, Devine, & Flaherty, 2005). DRUM uses a weighted sum of processor and communication cost for each node in a tree structure that represents the network topology. Different partitioning algorithms can be applied at different levels of the tree, for example fast geometric algorithms can be used to partition within a shared memory compute node where communication costs are negligible. Lan, Taylor, & Li (2006) used a hierarchical algorithm to balance the load between and within clusters for adaptive cosmological simulations. This deals to some extent with the hierarchical networks of computational grids without treating communication cost directly. Koenig & Kalé (2007) incorporated resource-aware load balancing into the Charm++ parallel programming language and runtime system. They used METIS to first partition into one subgraph per cluster, and then partition within each cluster. This means partitions must be recomputed from scratch at each load balancing step (“scratch-map repartitioning; see Schloegel, Karypis, & Kumar, 2003), which can have significant overhead, but the problem size they tested was quite small (3000 nodes).

### **Platform Modeling**

The load balancing algorithms described here employ a wide range of metrics to represent platform performance. Processor performance alone can be modeled using either benchmarks or monitoring of the application. More interestingly, different approaches have been taken to represent the total communication and computation cost and the network topology. For the former, terms representing communication and computation costs are either simply summed (MinEX, PART), or the terms are weighted in acknowledgment of the varying importance of the two terms for different applications (DRUM, PaGrid). It is not clear whether it is better to represent the network topology directly, or indirectly via a hierarchical approach. The former approach does have the advantage that it can deal with a more kinds of heterogeneous platforms. Finally, consideration of the importance of the number (sensitive to latency) and size of messages (sensitive to bandwidth) may be important (Aubanel & Wu, 2007).

### **Summary**

This review has revealed a wide range of solutions to the resource-aware load balancing problem, which are summarized in Table 1. What is evidently missing is a performance comparison between them, which it not possible at present, since they address applications with different characteristics. In order to compare them two things are needed: benchmark applications and simulators/emulators of grid computing systems. A consistent set of parallel benchmarks (of the type described in the Background section) is required in order to perform a fair comparison. However these benchmarks should not just be run on whatever heterogeneous system is at hand. For reproducible and comparable results they should be run on a standard set of simulated or emulated heterogeneous platforms. We know of no published benchmarks. However, some work

has been done on simulation (Liu & Chien, 2006) and on emulation (Koenig & Kalé, 2007; Canon & Jeannot, 2006).

**Table 1: Classification of resource-aware load balancing algorithms**

<b>Static</b>	<b>Partition and Map</b>
	<p><i>Recursive bipartition:</i> SCOTCH (Pellegrini &amp; Roman, 1996)</p> <p><i>Multilevel refinement:</i></p> <p>METIS (Karypis &amp; Kumar, 1998)</p> <p>JOSTLE (Walshaw &amp; Cross, 2001)</p> <p>PaGrid (Huang, Aubanel &amp; Bhavsar, 2003, 2006; Wanschoor &amp; Aubanel, 2004; Aubanel &amp; Wu, 2007)</p> <p><i>Neural network:</i> Adams &amp; Price (2004)</p> <p><i>Adaptation of METIS:</i> Moulitsas &amp; Karypis (2006)</p>
	<b>Hierarchical</b>
	<p><i>Simulated annealing:</i> PART (Chen &amp; Taylor, 2002)</p> <p><i>Using METIS:</i> Otero et al. (2005)</p>
<b>Dynamic</b>	<b>Processor Heterogeneity</b>
	<p><i>Octree, CFD:</i> Minyard &amp; Kallinderis (2000)</p> <p><i>Explicit FEM:</i> Rao (2006)</p> <p><i>Red-black SOR:</i> Dobber, Koole, &amp; van der Mei (2004)</p>
	<b>Network Heterogeneity</b>
	<p><i>AMR:</i> GrACE (Sinha &amp; Parashar, 2002)</p> <p><i>Multilevel refinement:</i> MinEX (Das, Harvey, &amp; Biswas, 2002)</p> <p><i>Multi-block CFD:</i> Petcu, Vizman, &amp; Paprzycki (2006)</p>
	<b>Hierarchical</b>
	<p><i>Tree-based model:</i> DRUM (Devine et al., 2005; Teresco, Devine, &amp; Flaherty, 2005)</p> <p><i>Adaptive Cosmological:</i> Lan, Taylor, &amp; Li (2006)</p> <p><i>Scratch-map using METIS:</i> Charm++ (Koenig &amp; Kalé, 2007)</p>

## **FUTURE TRENDS**

An important question that has not been addressed by the work surveyed here is whether it is better to select a subset of the processors of a computational grid at runtime instead of using them all. Moldable scheduling has been proposed to address this question for scheduling of homogeneous parallel applications (Cirne & Berman, 2003). This needs to be extended to heterogeneous platforms.

In dynamic load balancing there is a tradeoff between the benefit and cost of potentially complex redistribution strategies. This will likely be explored further, with future tools perhaps making available a choice of multiple strategies.

One important issue that has not been addressed here is fault tolerance. This is particularly important for high throughput platforms where member nodes may join and leave the grid frequently, but also for large high capacity grids, whose large size means that node failures are likely during the execution of an application. Solutions to this problem should recognize that MPI is the dominant programming paradigm in the HPC community and is likely to remain so for a while. Therefore solutions that require minimum alteration of existing code should be explored. Already, much work has been done on making MPI fault-tolerant, for example in OpenMPI (<http://www.open-mpi.org>).

Finally, much of the above work relies on the local nature of inter-task communication. Many new types of simulations use graphs with highly irregular connectivity, posing a challenge to graph partitioners. Techniques such as hypergraph partitioning are being explored to deal with such graphs (Devine et al., 2005).

## **CONCLUSION**

A rich set of solution techniques have been proposed for the problem of resource-aware load balancing of parallel applications, considering the static and dynamic heterogeneities of computational grids. In practice, the best approach will depend on the characteristics of the application and of the platform. These techniques need to be incorporated into software platforms (as in the case of Zoltan), and performance comparison of different approaches is necessary to characterize the balance between model sophistication, overhead, and solution quality.

## REFERENCES

Adams, J. R., & Price, C. C. (2004). Boltzmann algorithms to partition and map software for computational grids. *High Performance Grid Computing Workshop, in Proc. 18th International Parallel and Distributed Processing Symposium*.

Anderson, D. P. (2004). BOINC: A system for public-resource computing and storage. *Fifth IEEE/ACM International Workshop on Grid Computing*.

Attiya, G., & Hamam, Y. (2004). Two phase algorithm for load balancing in heterogeneous distributed systems. *12th Euromicro Conference on Parallel, Distributed and Network-Based Processing*, 434-439.

Aubanel, E., & Wu, X. (2007). Incorporating latency in heterogeneous graph partitioning. *Workshop on Parallel and Distributed Scientific and Engineering Computing (PDSEC), in Proc. 21st Intl. Parallel and Distributed Processing Symposium*.

Botadra, H., Cheng, Q., Prasad, S.K., Aubanel, E., and V. Bhavsar, V. (2007) iC2mpi: A Platform for Parallel Execution of Graph-Structured Iterative Computations. *Workshop on Parallel and Distributed Scientific and Engineering Computing (PDSEC), in Proc. 21st Intl. Parallel and Distributed Processing Symposium*, March 2007, Long Beach California.

Canon, L., & Jeannot, E. (2006). Wrekavoc: A tool for emulating heterogeneity. *Heterogeneous Computing Workshop (HCW), Proc. 20th Intl. Parallel and Distributed Processing Symposium*.

Cappuccio, R., Cattaneo, G., Erbacci, G., & Jocher, U. (April 2001). A parallel implementation of a cellular automata based model for coffee percolation. *Parallel Computing*, 27, 685-717(33).

Chen, J., & Taylor, V. E. (2002). Mesh partitioning for efficient use of distributed systems. *IEEE Transactions on Parallel and Distributed Systems*, 13(1), 67-79.

Chien, A. A. (2003). Architecture of a commercial enterprise desktop grid: The entropy system. In F. Berman, G.C. Fox, and A.J.G. Hey, *Grid computing: Making the Global Infrastructure a Reality*, John Wiley & Sons, Ltd.

Cirne, W., & Berman, F. (2003). When the herd is smart: Aggregate behavior in the selection of job request. *IEEE Transactions on Parallel and Distributed Systems*, 14(2), 181-192.

Das, S. K., Harvey, D. J., & Biswas, R. (2002). MinEX: A latency-tolerant dynamic partitioner for grid computing applications. *Future Generation Computer Systems*, 18(4), 477-489.

DeFanti, T. A., Foster, I., Papka, M. E., Stevens, R., & Kuhfuss, T. (1996). Overview of the I-way: Wide-area visual supercomputing. *International Journal of High Performance Computing Applications*, 10(2-3), 123-131.

Devine, K. D., Boman, E. G., Heaphy, R. T., Hendrickson, B. A., Teresco, J. D., Faik, J., et al. (2005). New challenges in dynamic load balancing. *Applied Numerical Mathematics*, 52(2-3), 133-152.

Dobber, M., Koole, G., & van der Mei, R. (2005). Dynamic load balancing experiments in a grid. *International Symposium on Cluster Computing and the Grid (CCGrid)*, 2 1063-1070 Vol. 2.

Dongarra, J., & Lastovetsky, A. (2006). An overview of heterogeneous high performance and grid computing. In B. Di Martino, J. Dongarra, A. Hoisie, L. Yang & H. Zima (Eds.), *Engineering the grid: Status and perspectives*, American Scientific Publishers.

- Foster, I. (1995). *Designing and Building Parallel Programs*, Addison Wesley.
- Foster, I., & Kesselman, C. (1999). *The grid: Blueprint for a new computing infrastructure*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.
- Grama, A., Karypis, G., Kumar, V., & Gupta, A. (2003). *Introduction to parallel computing* (2nd ed.), Addison-Wesley.
- Hendrickson, B. (1998). Graph partitioning and parallel solvers: Has the emperor no clothes? *Workshop on Parallel Algorithms for Irregularly Structured Problems, Lecture Notes in Computer Science Vol. 1457*, 218-225.
- Huang, S., Aubanel, E. E., & C., V. (2006). PaGrid: A mesh partitioner for computational grids. *Journal of Grid Computing*, 4(1), 71-88.
- Huang, S., Aubanel, E., & Bhavsar, V. C. (2003). Mesh partitioners for computational grids: A comparison. *International Conference on Computational Science and its Applications (LNCS 2267-2269)*, 60-68.
- Karypis, G., & Kumar, V. (1998). Multilevel k-way partitioning scheme for irregular graphs. *Journal of Parallel and Distributed Computing*, 48(1), 96-129.
- Koenig, G. A., & Kalé, L. V. (2007). Optimizing distributed application performance using dynamic grid topology-aware load balancing. 21<sup>st</sup>. *International Parallel and Distributed Processing Symposium*.
- Lan, Z., Taylor, V. E., & Li, Y. (2006). DistDLB: Improving cosmology SAMR simulations on distributed computing systems through hierarchical load balancing. *Journal of Parallel and Distributed Computing*, 66(5), 716-731.
- Li, Y. W., & Lan, Z. L. (2004). A survey of load balancing in grid computing. *First International Symposium on Computational and Information Science (Lecture Notes in Computer Science Vol. 3314)*, 3314 280-285.
- Liu, X., & Chien, A. A. (2006). Realistic large-scale online network simulation. *International Journal of High Performance Computing Applications*, 20(3), 383-399.
- Minyard, T., & Kallinderis, Y. (2000). Parallel load balancing for dynamic execution environments. *Computer Methods in Applied Mechanics and Engineering*, 189(4), 1295-1309.
- Moulitsas, I., & Karypis, G. (2006). Architecture aware partitioning algorithms. *Technical report No. TR 06-001*, University of Minnesota.
- Otero, B., Cela, J. M., Badia, R. M., & Labarta, J. (2005). Data distribution strategies for domain decomposition applications in grid environments. *6th International Conference on Algorithms and Architectures for Parallel Processing, ICA3PP (Lecture Notes in Computer Science Vol. 3719)*, 214-224.
- Pellegrini, F., & Roman, J. (1996). SCOTCH: A software package for static mapping by dual recursive bipartitioning of process and architecture graphs. *HPCN Europe 1996: Proceedings of the International Conference and Exhibition on High-Performance Computing and Networking*, 493-498.
- Petcu, D., Vizman, D., & Paprzycki, M. (2006). Heuristic Load Balancing for CFD Codes Executed in Heterogeneous Computing Environments. *Scalable Computing: Practice and Experience (SCPe)*, 7(2), 15-24.
- Rao, A.R.M. (2006). Explicit nonlinear dynamic finite element analysis on homogeneous/heterogeneous parallel computing environment. *Advances in Engineering Software*, 37(11), 701-720.
- Rotaru, T., & Năgeli, H. -. (2004). Dynamic load balancing by diffusion in heterogeneous systems. *Journal of Parallel and Distributed Computing*, 64(4), 481-497.

Schloegel, K., Karypis, G., & Kumar, V. (2003). Graph partitioning for high-performance scientific simulations. In J. Dongarra, et al. (Eds.), *The sourcebook of parallel computing*, Morgan Kaufmann.

Sinha, S., & Parashar, M. (2002). Adaptive system sensitive partitioning of AMR applications on heterogeneous clusters. *Cluster Computing*, 5(4), 343-352.

Stockinger, H. (2007). Defining the grid: A snapshot on the current view. *The Journal of Supercomputing*, 42(1), 3-17.

Teresco, J., Devine, K., & Flaherty, J. (2005). Partitioning and dynamic load balancing for the numerical solution of partial differential equations. In *Numerical solution of partial differential equations on parallel computers*, Springer Verlag.

Walshaw, C., & Cross, M. (2001). Multilevel mesh partitioning for heterogeneous communication networks. *Future Generation Computer Systems*, 17(5), 601-623.

Wanschoor, R., & Aubanel, E. (2004). Partitioning and mapping of mesh-based applications onto computational grids. *Fifth IEEE/ACM International Workshop on Grid Computing*, 2004, 156-162.

## Terms and Definitions

**Heterogeneous Computing** – Design and implementation of algorithms for computing platforms with heterogeneous characteristics, such as processor and network performance.

**Graph Partitioning** – In the context of parallel computing, the partitioning of a graph (representing concurrent execution of tasks and communication between tasks) into disjoint subgraphs, where the communication between subgraphs is minimized and the load is roughly balanced.

**Resource-Aware Load Balancing** – Assignment of tasks to a computational platform, taking into account its heterogeneous characteristics, with the objective of balancing the work assigned to processors. The tasks may be independent, or may have interdependencies described in a graph or digraph. In the latter case an additional objective is to minimize communication between processors. The assignment may be static, that is decided before tasks execute, or dynamic, that is tasks are assigned as conditions change.

**Static Load Balancing** – Assignment of tasks to a computational platform, before execution takes place, with the objective of balancing the work assigned to processors. When the tasks are interdependent an additional objective is to minimize communication between processors. This assignment is then fixed for the duration of the computation.

**Dynamic Load Balancing** – Assignment of tasks to a computational platform, with the objective of balancing the work assigned to processors. When the tasks are interdependent an additional objective is to minimize communication between processors. The load balance is monitored during execution of the application, and the assignment is modified if required, while trying to minimize the communication cost of migrating tasks.

**High Capacity Computational Grid** – A grid platform dedicated to high performance computing (HPC) applications and generally consisting of an aggregation of multiple HPC clusters. These grids are space-shared, that is processors only execute a single user's process(es) at a time. The clusters may be located in one organization, or they may be more spread out geographically, which can pose additional problems of high latencies.

**High Throughput Computational Grid** – A grid platform that aims to take advantage of spare compute cycles on processing nodes. These nodes may be dedicated to multiple users at the same time and hence are time-shared. These grids can consist of local or global networks of computers, and can be either within one organization or be simply a collection of individual computers on the Internet.