Asian Journal of Information Technology 5 (10): 1095-1103, 2006


# Load Balancing in Grid Computing

Belabbas Yagoubi, Hadj Tayeb Lilia and Halima Si Moussa
Department of Computer Science, University of Oran, Algeria

**Abstract:** Grid computing has recently emerged as popular platforms for deploying large-scale and resource-intensive applications. This kind of infrastructure raises challenging issues in many areas of computer science and especially in the area of distributed computing. One motivation of Grid computing is to aggregate the power of widely distributed resources and provide non-trivial services to users. To achieve this goal, large collaborative efforts are currently underway to provide the necessary software infrastructure. Resource management is an essential function provided at the service level of this software infrastructure. To improve the global throughput of these environments, workloads have to be evenly balanced among the available resources. Several load balancing strategies and algorithms have been proposed in this area. Most of them were developed in mind, assuming homogeneous set of sites linked with homogeneous and fast networks. However for computational grids we must address main new issues, like: heterogeneity, autonomy, dynamicity and so forth. This paper deals with a survey for grid load balancing problem. First, the essential aspects of load balancing system are overviewed to provide a global image of the load balancing process. Then specificities and challenges for grid are discussed and compared to traditional distributed systems. Finally, the state of the art of current research and some popular algorithms are outlined.

**Key words:** Distributed systems, grid computing, load balancing, workload

## INTRODUCTION

The current computational power demands and constraints of organizations have led to a new type of collaborative computing environment called Grid Computing[1]. A computational grid is an emerging computing infrastructure that enables effective access to high performance computing resources. End users and applications see this environment as a big virtual computing system. The systems connected together by a grid might be distributed globally, running on multiple hardware platforms, under different operating systems and owned by different organizations. While simultaneous resource allocations can be done manually with privileged accesses to the resources, such environments need certain resource management strategy and tools that can provide security and coordination of the resource allocations.

There are many potential advantages to using grid architectures, including the ability to simulate applications whose computational requirements exceed local resources and the reduction of job turnaround time through workload balancing across multiple computing facilities[2].

Resource management and load balancing are key grid services, where issues of local balancing represent a common concern for most grid infrastructure developers[3,4]. In fact, it would be inaccurate to say that the computing power of any system increases proportionally with the number of resources involved. Care should be taken so that some resources do not become overloaded and some others stay idle[5].

To improve the global throughput of grids software, computation requests have to be evenly balanced among the available resources. An important issue of such systems is the efficient assignment of tasks and utilization of resources, commonly referred to as load balancing problem and known to be NP complete[6]. It is essential to evenly distribute the workload among the available resources. In other words, it is desirable to prevent, if possible, the condition where one resource is overloaded with a large set of tasks to be serviced while another is lightly loaded or even idle. The main goal of this study is to highlight current research on load balancing problem for grid computing.

## GENERAL LOAD BALANCING PROBLEM

**Overview:** A typical distributed system will have a number of resources working independently with each other. Some of them are linked by communication channel and while some are not. Each resource possesses an initial load, which represents an amount of work to be performed

**Corresponding Author:** Belabbas Yagoubi, Department of Computer Science, University of Oran, Algeria

and each may have a different processing capacity. To minimize the time needed to perform all tasks, the workload has to be evenly distributed over all resources based on their processing speed. This is why *load balancing* is needed. The load balancing problem is closely related to scheduling and resource allocation. It is concerned by all techniques or methods allowing an evenly distribution of the workload among the available resources in a system[5].

In any practical distributed systems, the channels are of finite bandwidth and the processing units may be physically distant; Therefore, load balancing is also a decision making process of whether to allow tasks migration or not. The essential objective of a load balancing consists primarily to optimize the average response time of applications, which often mean to maintain the workload proportionally equivalent on the whole resources of a system. Load balancing is usually described in the literature[7], as either load balancing or load sharing. These terms are often used interchangeably, but can also attract quite distinct definitions.

**Load sharing:** This is the coarsest form of load distribution. Load may only be placed on idle resources and can be viewed as binary, where a resource is either idle or busy.

**Load balancing:** Where load sharing is the coarsest form of load distribution, load balancing is the finest. Load balancing attempts to ensure that the workload on each resource is within a small degree, or balance criterion, of the workload present on every other resource in the system.

**Load levelling:** Load levelling introduces a third term to describe the middle ground between the two extremes of load sharing and load balancing. Rather than trying to obtain a strictly even distribution of load across all resources, or simply utilising idle resources, load levelling seeks to avoid congestion on any one resource.

Another issue related to load balancing is that a computing job may not arbitrarily divisible leading to certain constraints in dividing tasks. Each job consists of several tasks and each of those tasks can have different execution times. Also, the load on each resource as well as on the network can vary from time to time based on the workload brought about by the users. The resource capacity may be different from each other in architecture, operation system, CPU speed, memory size and available disk space. The load balancing problem also needs to consider fault-tolerance and fault-recovery. With all these factors taken into account, load balancing can be generalized into four basic steps[8]

- Monitoring resource load and state;
- Exchanging load and state information between resources (sites, nodes);
- Calculating the new work distribution;
- Actual data movement.

The problem of load balancing across a network of available resources has been discussed in distributed systems literature for more than two decades. Various load distribution algorithms have been proposed, implemented and classified in various studies[9,6].

**Policies and mechanisms:** There are two basic issues in the design of load balancing system[10]. The policy issue is the set of choices that are made to balance the load (which tasks should be executed remotely and where). The mechanism issue carries out the physical facilities to be used for task remote execution and provides any information required by the policies. The division of policy and mechanism can be continued, breaking any load balancing scheme into a set of distinct but interdependent components.

Figure1 illustrates a suitable decomposition with each leaf representing a distinct component of a load distribution scheme. The emphasis is on the components of the policy and the provision of information to the policy. The following list briefly explains each component identified by a leaf of the tree.

**Candidate selection policy:** The candidate policy selects the tasks to be distributed. This can be for reasons ranging from insufficient service on the current resource, to the reduction of communication paths.

**Load metric mechanism:** The load metric is the representation used to describe the load on a resource. This determines the type of information that makes a

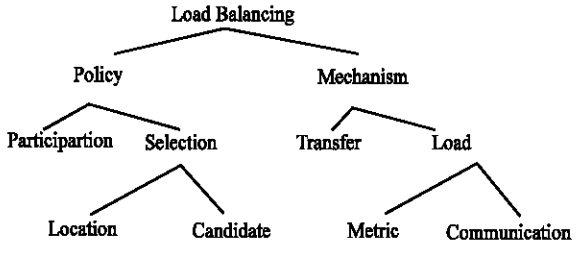


Fig. 1: The components of load balancing system

load index (queue CPU length, memory size) and the way such information is communicated to other loaders (broadcasting, focused addressing, polling).

**Load communication mechanism:** Load communication defines the method by which information, such as the load on a resource, is communicated between the resource and the load distribution policies/mechanisms. The load communication policy can also include the communication between cooperating distributed policies.

**Transfer mechanism:** The transfer mechanism is the physical medium for transferring tasks between resources. This leaf can be further expanded to include branches for initial placement and process migration.

Although the leaves in the tree structure are distinct, the individual components are not independent. The choice of load metric for example, affects the location selection and possibly the candidate selection as well. If this structure were used to construct a load balancing mechanism bottom up, by selecting components and joining them together, or to identify and then replace a single component, care must be taken to ensure that each part of the system is capable of providing the functionality required by the rest of the system. As an example, consider the mismatch in a system where a candidate selection policy required migration to reduce communication paths, but was provided with initial placement as the transfer mechanism.

## TAXONOMY OF LOAD BALANCING POLICIES

There has been an extensive research in the development of the appropriate load balancing policy. Approaches of scheduling (load balancing) presented in the literature are so numerous that it is practically impossible to cover each one of them in all its details. The problem description knew a proliferation of the terminologies, sometimes even contradictory, making difficult the qualitative analysis of the various methods suggested. Thus, it had become necessary to have a taxonomy which makes it possible to standardize the terminologies for a better description of these approaches and their comparison. In[11], Casavant et al propose a largely adopted taxonomy, because very complete, for scheduling and load balancing algorithms in general-purpose parallel and distributed computing systems.

The organization of the different load balancing schemes is shown in Fig. 2. From the top to the bottom, this structure can be identified as what follows.

**Static versus dynamic:** Static load balancing, also known as deterministic distribution, assigns a given job to a fixed resource. Every time the system is restarted, the same

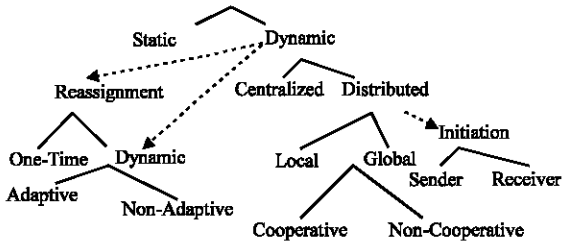


Fig. 2: Hierarchical taxonomy for load balancing policies

binding task resource (allocation of a task to the same resource) is used without considering changes that may occur during the system's lifetime. Moreover, static load balancing may also characterize the strategy used at runtime, in the sense that it may not result in the same task resource assignment, but assigns the newly arrived jobs in a sequential or fixed fashion. For example, using a simple static strategy, jobs can be assigned to resources in a round-robin fashion so that each resource executes approximately the same number of tasks.

In this mode, every task comprising the application is assigned once to a resource. Thus, the placement of an application is static and a firm estimate of the cost of the computation can be made in advance of the actual execution. One of the major benefits of the static model is that it is easier to implement.

Dynamic load balancing takes into account that the system parameters may not be known beforehand and therefore using a fixed or static scheme will eventually produce poor results. A dynamic strategy is usually executed several times and may reassign a previously scheduled task to a new resource based on the current dynamics of the system environment.

This model is usually applied when it is difficult to estimate the costs of applications are coming online dynamically. It has two major components[12]: system state estimation (other than cost estimation in static case) and decision making. System state estimation involves collecting state information throughout the system and constructing an estimate. On the basis of the estimate, decisions are made to assign a task to a selected resource. Since the cost for an assignment is not available, a natural way to keep the whole system health is balancing the loads of all resources. The advantage of dynamic load balancing over static is that the system need not be aware of the run-time behaviour of the application before execution.

**Distributed vs. centralized:** In dynamic load balancing, the responsibility for making global decisions may lie with one centralized location, or be shared by multiple distributed locations. The centralized strategy has the

advantage of ease of implementation, but suffers from the lack of scalability, fault tolerance and the possibility of becoming a performance bottleneck.

In distributed load balancing, the state information is distributed among the nodes that are responsible in managing their own resources or allocating tasks residing in their queues to other nodes. In some cases, the scheme allows idle resources to assign tasks to themselves at runtime by accessing a shared global queue. Note that failures occurring at a particular node will remain localized and may not affect the global operation of the system.

Another scheme that fits between the two types above is the hierarchical one where selected nodes are responsible for providing tasks to a group of nodes. The nodes are arranged in a tree and the selected nodes are roots of the sub tree domains.

A property of this algorithm is that it uses a smart search strategy to find partner nodes to which tasks can migrate. It also overlaps this decision making process with the actual execution of ready jobs, thereby saving precious resource cycles.

**Local vs. global:** Local and global load balancing fall under the distributed scheme since a centralized scheme should always act globally. In a local load balancing, each resource polls other resources in its neighbourhood and uses this local information to decide upon a load transfer. This local neighbourhood is usually denoted as the migration space. The primary objective is to minimize remote communication as well as efficiently balance the load on the resources. However, in a global balancing scheme, global information of all or part of the system is used to initiate the load balancing. This scheme requires a considerable amount of information to be exchanged in the system which may affect its scalability.

**Cooperative vs. Non-cooperative:** If a distributed load balancing mode is adopted, the next issue that should be considered is whether the nodes involved in job balancing are working cooperatively or independently (non-cooperatively). In the non-cooperative case, individual loaders act alone as autonomous entities and arrive at decisions regarding their own optimum objects independent of the effects of the decision on the rest of system.

**Adaptive vs. Non adaptive:** Adaptive and non-adaptive schemes are part of the dynamic load-balancing policies. In an adaptive scheme, scheduled decisions take into consideration past and current system performance and are affected by previous decisions or changes in the environment. If one (or more parameters) does not correlate to the program performance, it is weighted less next time. In the non-adaptive scheme, parameters used in balancing remain the same regardless of system's past behaviour. An example would be a policy that always weighs its inputs the same regardless of the history of the system behaviour.

Confusion may arise between in distinguishing dynamic balancing and adaptive balancing. Whereas a dynamic solution takes environmental inputs into account when making its decision, an adaptive solution (which is also dynamic) takes environmental stimuli into account to modify the load balancing policy itself[13].

**One-time assignment vs dynamic reassignment:** In this classification, the entities to be balanced are considered. The one-time assignment of a task may be dynamically done but once it is scheduled to a given Resource, it can never be migrated to another one[14]. On the other hand, in the dynamic reassignment process, jobs can migrate from one node to another even after the initial placement is made. A negative aspect of this scheme is that tasks may endlessly circulate about the system without making much progress.

**Sender/receiver/symmetrically initiated balancing:** Techniques of balancing tasks in distributed systems have been divided mainly as sender-initiated, receiver-initiated and symmetrically-initiated[15,3]. In sender-initiated models, the overloaded nodes transfer one or more of their tasks to more under-loaded nodes. In receiver-initiated schemes, under-loaded nodes request tasks to be sent to them from nodes with higher loads. In symmetric approach, both the under-loaded as well as the loaded nodes can initiate load transfers.

## LOAD BALANCING IN GRID COMPUTING

A Computational Grid is a hardware and software infrastructure that provides dependable, consistent, pervasive and inexpensive access to high-end computational capabilities[16]. It is a shared environment implemented via the deployment of a persistent, standards-based service infrastructure that supports the creation of and resource sharing within, distributed communities. Resources can be computers, storage space, instruments, software applications and data, all connected through the Internet and a middleware software layer that provides basic services for security, monitoring, resource management and so forth. Resources owned by various administrative organizations are shared under locally defined policies that specify what is shared, who is allowed to access what and under what conditions. The real and specific problem that underlies the Grid concept is coordinated resource sharing and problem solving in dynamic, multi-institutional virtual organizations[17].

**The grid load balancing process:** A Grid is a system of high diversity, which is rendered by various applications, middleware components and resources. In[18], Schoft generalize a load balancing process in the Grid into three stages: information collection, resource selection and tasks mapping.

**Information collection:** Information Collection is the basis for providing current state information of the resources. It should be performed during the whole course of system running. A load balancing system can either construct its own information collection infrastructure, or employ existing information service systems, enabled by middleware's. It is desired that the overhead introduced by the process of information collection is as small as possible.

**Resource selection:** In principle, resource selection is performed in two steps. In the first step, the initial filtering is done with the goal of identifying a list of authorized resources that is available to a given application. Possibly, the initial list of authorized resources can be further refined by filtering according to the coarse-grained application requirements, such as hardware platform, operating system, minimum memory and disk space.

In the second step, those resources are aggregated into small collections such that each collection is expected to provide performance desired by the given application. The number of ways that the resources could be aggregated would be extremely large when the number of feasible resource is large. To over-come the complexity, different heuristics may be introduced.

**Tasks mapping:** The third phase involves mapping the given set of tasks onto a set of aggregated resources including both the computational resources and network resources. This is a well-know NP-complete problem and various heuristics may be used to reach a near-optimal solution. The effort of mapping in conjunction of resource selection produces a set of candidate submissions. Once the set of candidate submissions is ready, the balancer starts to select the best submission subject to the performance goal, based on mechanisms provided by the performance model.

## CHALLENGES OF LOAD BALANCING IN GRID COMPUTING

Load balancing systems for traditional distributed environments do not work in Grid environments because the two classes of environments are radically distinct. Load balancing in Grid environments is significantly complicated by the unique characteristics of Grids.

Traditional load balancing models generally produce poor Grid balancing in practice. The reason can be found by going through the assumptions underlying traditional systems[13]:

- All resources reside within a single administrative domain.
- To provide a single system image, the balancer controls all of the resources.
- The resource pool is invariant.
- Contention caused by incoming applications can be managed by the balancer according to some policies, so that its impact on the performance that the site can provide to each application can be well predicted.
- Computations and their data reside in the same site or data staging is a highly predictable process, usually from a predetermined source to a predetermined destination, which can be viewed as a constant overhead.

Unfortunately, all these assumptions do not hold in Grid circumstances. In Grid computing, many unique characteristics make the design of load balancing algorithms more challenging[8], as explained in what follows.

**Resource heterogeneity:** A computational Grid mainly has two categories of resources: networks and computational resources. Heterogeneity exists in both of the two categories of resources. First, networks used to interconnect these computational resources may differ significantly in terms of their bandwidth and communication protocols. A wide-area Grid may have to utilize the best-effort services provided by the Internet. Second, computational resources are usually heterogeneous in that these resources may have different hardware, such as instruction set, computer architectures, number of resource, physical memory size, CPU speed and so on and also different software, such as different operating systems, file systems, cluster management software and so on. The heterogeneity results in differing capability of processing jobs. Resources with different capacity could not be considered uniformly. An adequate load balancing system should address the heterogeneity and further leverage different computing power of diverse resources.

**Site autonomy:** Typically a Grid may comprise multiple administrative domains. Each domain shares a common security and management policy. Each domain usually authorizes a group of users to use the resources in the

domain. Thus applications from non authorized users should not be eligible to run on the resources in some specific domains. Further more, a site is an autonomous computational entity. A shared site will result in many problems. It usually has its own scheduling policy, which complicates the prediction of a job on the site. A single overall *Performance goal* is not feasible for a Grid system since each site has its own performance goal and scheduling decision is made independently of other sites according to its own performance goal.

*Local priority* is another important issue. Each site within the Grid has its own scheduling policy. Certain classes of jobs have higher priority only on certain specific resources. For example, it can be expected that local jobs will be assigned higher priorities such that local jobs will be better served on the local resources.

Most traditional load balancers are designed with the assumption of having complete control of the underlying resources. Under this assumption, the load balancer has adequate information of resources and therefore effective load balancing is much easier to obtain. But in Grid environments, the Grid load balancer has only limited control over the resources. Site autonomy greatly complicates the design of effective Grid load balancing.

**Resource non-dedication:** Because of non dedication of resources, resource usage c*ontention* is a major issue. Competition may exist in both computational resources and interconnection networks. Due to the non dedication of resources, a resource may join multiple Grids simultaneously. The workloads from both local users and other Grids share the resource concurrently. The underlying interconnection network is shared as well. One consequence of contention is that behaviour and performance can vary over time. For example, in wide area networks using the Internet Protocol suite, network characteristics such as latency and bandwidth may be varying over time. Under such an environment, designing an accurate performance model is extremely difficult.

Contention is addressed by assessing the fraction of available resources dynamically and using this information to predict the fraction available at the time of application to be scheduled.

**Application diversity:** The problem arises because the Grid applications are from a wide range of users, each having its own special requirements. For example, some applications may require sequential execution, some applications may consist of a set of independent jobs and others may consist of a set of dependent jobs. In this context, building a general-purpose load balancing system seems extremely difficult. An adequate load balancing system should be able to handle a variety of applications.

**Dynamic behaviour:** In traditional distributed computing environments, such as a cluster, the pool of resources is assumed to be fixed or stable. In a Grid environment, dynamics exists in both the networks and computational resources. First, a network shared by many parities cannot provide guaranteed bandwidth. This is particularly true when wide-area networks such as the Internet are involved. Second, both the availability and capability of computational resources will exhibit dynamic behaviour. On one hand new resources may join the Grid and on the other hand, some resources may become unavailable due do problems such as network failure. The capability of resources may vary overtime due to the contention among many parties who share the resources. An adequate load balancing should adapt to such dynamic behaviour. After a new resource joins the Grid, the load balancing should be able to detect it automatically and leverage the new resource in the later balancing decision making. When a computational resource becomes unavailable resulting from an unexceptional failure, mechanisms, such as checkpointing or rebalancing, should be taken to guarantee the reliability of Grid systems.

**Resource selection and computation-data separation:** In traditional systems, executable codes of applications and input/output data are usually in the same site, or the input sources and output destinations are determined before the application is submitted. Thus the cost for data staging can be neglected or the cost is a constant determined before execution and load balancing algorithms need not consider it. But in a Grid which consists of a large number of heterogeneous computing sites (from supercomputers to desktops) and storage sites connected via wide area networks, the computation sites of an application are selected by the Grid load balancer according to resource status and certain performance models. Additionally, in a Grid, the communication bandwidth of the underlying network is limited and shared by a host of background loads, so the inter domain communication cost cannot be neglected. Further, many Grid applications are data intensive, so the data staging cost is considerable. This situation brings about the computation data separation problem: the advantage brought by selecting a computational resource that can provide low computational cost may be neutralized by its high access cost to the storage site.

These challenges pose significant obstacles on the problem of designing an efficient and effective load balancing system for Grid environments. Some problems resulting from the above are not solved successfully yet and still open research issues. As a result, new load balancing frame work must be developed for Grids, which should reflect the unique characteristics of Grid systems.

## LOAD BALANCING ALGORITHMS IN GRID COMPUTING

It is well known that the complexity of a general load balancing problem is NP-Complete. As mentioned the load balancing problem becomes more challenging because of some unique characteristics belonging to Grid computing. To date, there have been a number of exciting initial efforts at developing load balancing systems for Grid environments. It is often difficult to make comparisons between distinct efforts because each load balancing is usually developed for a particular system environment or particular greedy application, with different assumptions and constraints. We attempt to outline the methodologies adopted and ideas behind popular load balancing system.

**Adaptive load balancing algorithms[4]:** In computational grids, applications need to simultaneously tap the computational power of multiple, dynamically available sites. The crux of designing load balancing grid environments stems exactly from the dynamic availability of compute cycles: grid programming environments need to be both portable to run on as many sites as possible and they need to be flexible to cope with different network protocols and dynamically changing groups of heterogeneous compute nodes. In others words, an adaptive solution to the load balancing problem is one in which the algorithms and parameters used to make load balancing decisions change dynamically according to the previous, current and/or future resource status[5,9]. The demand for load balancing adaptation comes from three points: the heterogeneity of candidate resources, the dynamism of resource performance and the diversity of applications. We can find three kinds of adaptation.

**Resource adaptation:** Because of resource heterogeneity and application diversity, discovering available resources and selecting an application appropriate subset of those resources are very important to achieve high performance or reduce the cost. [19] show how the selection of a data storage site affects the network transmission delay[20], propose a resource selection algorithm in which available resources are grouped first into disjoint subsets according to the network delays between the subsets. Then, inside each subset, resources are ranked according to their memory size and computational power. Finally, an appropriately sized resource group is selected from the sorted lists. The upper bound for this exhaustive search procedure is given and claimed acceptable in the computational Grid circumstance.

**Dynamic performance adaptation:** The adaptation to the dynamic performance of resources is mainly exhibited as:

- Changing load balancing policies[9] (switching between static load balancing algorithms which use predicted resource information and dynamic ones which balance the static load balancing results),
- Workload distributing according to application specific performance models and finding a proper number of resources to be used[21].

Applications to which these adaptive strategies are applied usually adopt some kind of divide-and-conquer approach to solve a certain problem[9].

**Application adaptation:** Aggarwall *et al*.,[22] consider that applications in Grid computing often meet, namely, resource reservation and develop a generalized Grid load balancing algorithm that can efficiently assign jobs having arbitrary inter-dependency constraints and arbitrary processing durations to resources having prior reservations. Their algorithm also takes into account arbitrary delays in transfer of data from the parent tasks to the child tasks. In[23], Wu et al give an algorithm adaptive to indivisible single sequential jobs, jobs that can be partitioned into independent parallel tasks and jobs that have a set of indivisible tasks. When prediction error of the system utilization is reaching a threshold, the scheduler will try to reallocate tasks.

## GENETIC ALGORITHMS

The classic approach is to use algorithms, which try to solve the load balancing problem by analyzing the current state of the system. This way has some disadvantages like the need of medium or large computing power and due to this problem the scalability is poor on larger systems.

To avoid this drawback, others approaches were proposed. One of these has been introduced analogies from natural phenomena to form powerful heuristics, which have proven to be highly successful. Some of the common characteristics of Nature's heuristics are the close resemblance to a phenomenon existing in nature, namely, non-determinism, the implicit presence of a parallel structure and adaptability[24,25]. We can find three basic heuristics implied by Nature for Grid scheduling, namely, Genetic Algorithm (GA), Simulated Annealing (SA) and Tabu Search (TS).

For its simplicity, GA is the most popular Nature's heuristic used in algorithms for optimization problems.

GA is an evolutionary technique for large space search. The general procedure of GA search is as follows:

**Population generation:** A population is a set of chromosomes and each represents a possible solution, which is a mapping sequence between tasks and machines.

**Chromosome evaluation:** Each chromosome is associated with a fitness value, which is the makespan of the task-machine mapping this chromosome represents. The goal of GA search is to find the chromosome with optimal fitness value.

**Crossover and mutation operation:** Crossover operation selects a random pair of chromosomes and chooses a random point in the first chromosome. For the sections of both chromosomes from that point to the end of each chromosome, crossover exchanges machine assignments between corresponding tasks. Mutation randomly selects a chromosome, then randomly selects a task within the chromosome and randomly reassigns it to a new machine.

Finally, the chromosomes from this modified population are evaluated again. This completes one iteration of the GA. The GA stops when a predefined number of evolutions is reached or all chromosomes converge to the same mapping.

## COGNITIVE ALGORITHMS

The other approach is the one, which try to estimate, some how, the future system state in order to propose a more stable solution[26]. This seems to be the future in this domain because the classical approach can drive to dramatic increase of communication into the computer network. In another words the achieved Speed up is very low or worst. This is happening due to the workstation user comportment, which can decide to change his computing needs at any moment. For this reason remote tasks are unacceptably delayed so from the genitor point of view those can be considered as dead and must be computed locally or sent to other low charged station in the net.

One way of estimating the workstations load is by using a statistical approach such as load functions resulted from repeated measurements. These functions often have a Gaussian aspect, like many other models of natural processes. But these methods ignore, as we previous mentioned, the very cause of the load of a workstation, which is the behavior of the individual user. In the last decades, the progress in cognitive science, especially in cognitive psychology, made possible for the development of behavioral models that can estimate, to a higher or lower degree of precision, the way an individual may act under certain circumstances. While no one can challenge the immense variety of the human nature, it is evident that people tend to act rationally in controlled environments and therefore their behavior is not totally random. This assumption leads to an attempt to discover some general rules that may determine (or at least approximate) the conduct of a person in simple, repeating situations. The combined effect of these rules represents the behavior of the individual.

## CONCLUSION

The goal of this survey is to discuses issues and methods deployed in load balancing systems for Grid environments. Although load balancing in parallel and distributed systems has been intensively studied, new challenges in Grid environments still make it an interesting topic and many research projects are underway. Through our survey on current load balancing algorithms working in the Grid computing, we can claim that load balancing for traditional systems cannot be applied to the Grid environments. On one hand, the Grid environments exhibit heterogeneous and dynamic characteristics, which distributed system environments do not have. On the other hand, Grid environments are intended to execute much diverse applications compared to the traditional environments.

The research topic of Grid load balancing is quite young. No very successful load balancing systems has been proposed and reported and a few proposed systems have many limitations. Hence, with the development and popularity of Grid computing, much work must be done in order to enable grid computing to be a real platform delivering high performance services.

In addition to enhancements to classic load balancing algorithms, new methodologies are applied, such as the Grid economic models and Nature's heuristics.

## REFERENCES

1. Foster, I., 2002. The grid: A new infrastructure for 21st Century Science. Hysics Today, 55: 42-47.
2. Foster, I. and C. Kesselman, 1998. The grid: blueprint for a new computing infrastructure, Morgan Kaufmann.
3. Arora, M., S.K. Das and R. Biswas, 2002. A decentralized scheduling and load balancing algorithm for heterogeneous grid environments. In Proc. of International Conference on Parallel Processing Workshops (ICPPW'02), Vancouver, British Columbia Canada, pp: 499-505.
4. Fangpeng Dong and G. Akl Selim, 2006. Scheduling algorithms for grid computing: state of the art and open problems. Technical Report No. 2006-504, School of Computing, Queen's University Kingston, Ontario.
5. Kabalan, K.Y., W.W. Smar and J.Y. Hakimian, 2002. Adaptive load sharing in heterogeneous systems: Policies, modifications and simulation. Intl. J. SIMULATION, 3: 89-100.

6. Xu, C.Z. and F.C.M. Lau, 1997. Load balancing in parallel computers: Theory and practice, Kluwer, Boston, MA.
7. Kristian Paul Bubendorfer, 1996. Resource based policies for load distribution. Master thesis, Victoria University of Wellington.
8. Zhu, Y., 2003. A survey on grid scheduling systems. Technical report, Department of Computer Science, Hong Kong University of Science and Technology.
9. Nieuwpoort, R.V., T. Kielmann and H.E. Bal, 2001. Efficient load balancing for wide-area divide-and-conquer applications. In Proc. Eighth ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP'01), Snowbird, UT, pp: 34-43.
10. Ferrari, D. and S. Zhou, 1987. An empirical investigation of load indices for load balancing applications. Proc. of 12th International Symposium on Computer Performance Modeling Measurement and Evaluation, pp: 515-528.
11. Casavant, T.L. and J.G. Kuhl, 1994. A taxonomy of scheduling in general purpose distributed computing systems. IEEE Trans. Software Engin., 14: 141-153.
12. Rotithor, H.G., 1994. Taxonomy of dynamic task scheduling schemes in distributed computing systems. In IEE Proc. on Computer and Digital Techniques, 141: 1-10.
13. Berman, F., 1998. High-performance schedulers, chapter in the grid: blueprint for a future computing infrastructure, edited by Foster I. and Kesselman C., Morgan Kaufmann Publishers.
14. Hamscher, V., U. Schwiegelshohn, A. Streit and R. Yahyapour, 2000. Evaluation of job-scheduling strategies for grid computing. GRID, pp: 191-202.
15. Shan, H., L. Oliker, R. Biswas and W. Smith, 2004. Scheduling in heterogeneous grid environments: The effects of data migration. In Proc. of ADCOM2004: International Conference on Advanced Computing and Communication, India.
16. Foster, I. and C. Kesselman C. (Eds.), 1999. The grid: blueprint for a future computing infrastructure, Morgan Kaufmann Publishers, USA.
17. Foster, I., C. Kesselman and S. Tuecke, 2001. The anatomy of the grid: Enabling scalable virtual organizations. In the Intl. J. Supercomputer Applications, 15: 200-220.
18. Schopf, J.,2001. Ten actions when superscheduling. Document of scheduling working group, global grid forum, http://www.ggf.org/documents/GFD.4.pdf.
19. Su, A., F. Berman, R. Wolski and M. Mills Strout, 1999. Using apples to schedule simple SARA on the computational grid. In International J. of High Performance Computing Applications, 13: 253-262.
20. Dail, H., H. Casanova and F. Berman, 2002. A decoupled scheduling approach for the GrADS environment. In Proc. 2002 ACM/IEEE conference on Supercomputing, Baltimore, Maryland USA, pp: 1-14.
21. Casanova, H., M. Kim, J.S. Plank and J.J. Dongarra, 1999. Adaptive scheduling for task farming with grid middleware. In the International J. High Performance Computing Applications, pp: 231-240.
22. Aggarwal, A.K. and R.D. Kent, 2005. An adaptive generalized scheduler for grid applications. In Proc. of the 19th Annual Intl. Symposium on High Performance Comput. Sys. Applications (HPCS'05), Guelph, Ontario Canada, pp: 15-18.
23. Wu, M. and X. Sun, 2003. A general self-adaptive task scheduling system for non-dedicated heterogeneous computing. In Proc. of IEEE International Conference on Cluster Computing (CLUSTER'03), Hong Kong, pp: 354-361.
24. Abraham, A., R. Buyya and B. Nath, 2000. Nature's Heuristics for scheduling jobs on computational grids. In Proc. of 8th IEEE Intl. Conf. Adv. Comput. Communicat. (ADCOM 2000), Cochin, India, pp: 45-52.
25. Alaoui, Frieder and El-Ghazawi, 1999. A parallel genetic algorithm for task mapping on parallel machines. Job scheduling strategies for parallel processing: {IPPS} '95 Workshop, LNCS.
26. Florin Leon, Dan Gâlea and Mihai Horia Zaharia, 2002. Load balancing in distributed systems using cognitive behavioral models. Bulletin of Technical University of Iaşi, tome XLVIII (LII), fasc. 1-4, pp: 119-124.