

A Survey of Load Balancing In High-Performance Distributed Computing Systems

S. F. El-Zoghdy\textsuperscript{1,3}, and S. Ghoniemy\textsuperscript{2}

\textsuperscript{1}Information Technology Dep., College of Computers & Information Technology, Taif University, Taif, KSA, Alzoghdy@tu.edu.sa
\textsuperscript{2}Computer Engineering Dep., College of Computers & Information Technology, Taif University, Taif, KSA, Ghoniemy@tu.edu.sa
\textsuperscript{3}Mathematics & Computer Science Dep., Faculty of Science, Menoufia University, Shebin El-Koom, Egypt.

Abstract—With the rapid development of high-speed wide-area networks and powerful yet low-cost computational resources, high performance distributed computing systems (HPDCS) came into being and are currently used. One of the main motivations of these systems is to aggregate the power of widely distributed resources, and provide non-trivial services to users. To achieve this goal, efficiently utilizing available system resources becomes an important research issue in these systems. Utilizing available system resources can be done by evenly redistributing system workload on the available resources which is known as load balancing. This paper provides a review of the subject mainly from the perspective of HPDCS, and their load balancing algorithms. In this review, the challenges for these systems and their load balancing algorithms are identified. First, the HPDCS are classified into three broad categories, namely: (a) cluster, (b) grid, and (c) cloud systems, and are briefly introduced to provide an intuitive image of these systems and their important role in our recent life. Then a comprehensive discussion of widely used load balancing algorithms deployed in HPDCS environments is presented. Finally, a classification of these algorithms is done from different points of view, such as static vs. dynamic, centralized vs. decentralized, and finally cooperative vs. non-cooperative algorithms.

Keywords—Distributed Computing Systems; Resource Management; Load Balancing; Performance Evaluation, and Improvement.

1. INTRODUCTION

Distributed systems refer to computer networks where individual computers are physically distributed within some geographical area. The computers interact with each other in order to achieve a common goal. A computer program that runs in a distributed system is called a \textit{distributed program}, and \textit{distributed programming} is the process of writing such programs [1]. \textit{Distributed computing} refers to the use of distributed systems to solve computational problems. A distributed computing system ties together the power of large number of resources distributed across a network [2].

In distributed computing, a problem is divided into many tasks, each of which is solved by one computer. The use of concurrent processes that communicate by message-passing has its roots in operating system architectures studied in the 1960s [10]. The first widespread distributed systems were local-area networks such as Ethernet, which was invented in the 1970s [9]. ARPANET, the predecessor of the Internet, was introduced in the late 1960s, and ARPANET e-mail was invented in the early 1970s. E-mail became the most successful application of ARPANET [1] and it is probably the earliest example of a large-scale distributed application. In addition to ARPANET, and its successor, the Internet, other early worldwide computer networks included Usenet and FidoNet from 1980s, both of which were used to support distributed discussion systems.

The study of distributed computing became its own branch of computer science in the late 1970s and early 1980s. The first conference in the field, Symposium on Principles of Distributed Computing (PODC), dates back to 1982, and its European counterpart International Symposium on Distributed Computing (DISC) was first held in 1985.

There are many reasons that encourage the use of distributed systems including:

a. \textbf{Resource sharing}: If a number of different computers are connected to each other via a network, then a user at any of these computers is able to use the resources available at the other computers. In general, resource sharing in distributed computer systems provides mechanisms for sharing and using remote hardware and software resources.

b. \textbf{Performance Improvement}: If the problem to be solved is partitioned into a number of independent sub-problems that can run concurrently, then the availability of distributed computing systems allow us to distribute the computation among various computing nodes to run it concurrently.

c. \textbf{Application Nature}: The nature of parallel and distributed applications suggests the use of a communication network that connects several computers. Such networks are necessary for producing data that are required for the execution of tasks on remote resources. Also, most of the parallel and distributed applications have multiple processes that run concurrently on many computers (nodes) communicating over a high-speed interconnect. The use of high performance distributed systems for parallel and distributed applications is beneficial as compared to a single Central Processing Unit (CPU) machine for many practical reasons. The
ability of services distributed in a wide network is low-cost and makes the whole system scalable and adapted to achieve the desired level of the performance efficiency.

**d. Reliability:** The reliability of the distributed system is higher than a monolithic single processor machine. A single failure of one network node in a distributed environment does not stop the whole process as compared to a single CPU resource. Moreover, a distributed system may be easier to expand and manage than a monolithic uni-processor system.

**e. Information Exchanging:** when a number of computers are connected to each other, the users at different computers have the opportunity to exchange information.

Generally, the main motivations for the users of distributed computing systems are performance improvement, scalability, reliability, information sharing, and information exchange [3,4]. A distributed system is mainly heterogeneous in nature in the sense of the computing nodes, network topology, communication medium, operating system etc. [13,14]. Improving performance is one of the main important challenges in distributed computing systems. As it is shown in Fig. 1, random arrival of tasks to the computing nodes in a distributed system may lead to a situation where some computing nodes become heavily loaded while others in the system may be lightly loaded or even idle. It is therefore desirable to transfer some tasks from the heavily loaded computers to the idle or lightly loaded ones aiming to efficiently utilize the available computing resources and hence improve the hole system performance. The process of load redistribution is known as load balancing [4-6]. There has been extensive research in the development of load balancing Strategies [2-8, 13-18, 33-37].

In this research paper we present a brief introduction to high performance distributed computing systems and a general survey of load balancing strategies in these systems.

The rest of the paper is organized as follows. Section 2 presents an overview for the current high performance distributed computing systems. Section 3 gives a survey of load balancing strategies in distributed computing systems. Finally section 4 summarizes the paper.

## 2. HIGH PERFORMANCE DISTRIBUTED COMPUTING SYSTEMS (HPDCS)

In this section, we present the three main categories of HPDCS namely: Cluster computing systems, Grid computing systems, and Cloud computing systems.

### 2.1 Cluster Computing Systems (CCS)

A modern cluster computing system is made up of a set of computer nodes (whole computers work together as a unified computing resource that can create the illusion of being one machine having parallel processing) that are usually restricted to a single switch or group of interconnected switches within a single virtual local-area network (VLAN). Each node may have different architecture specifications (single processor machine, symmetric multiprocessor system, etc.) and access to various types of storage devices. The underlying network is a dedicated network made up of high-speed and low-latency system of switches with a single or multi-level hierarchic internal structure. In addition to executing compute-intensive applications, cluster systems are also used for replicated storage and backup servers that provide essential fault tolerance and reliability for critical parallel applications [2].

The main goal of cluster computing is to design an efficient computing platform that uses a group of commodity computing resources integrated through hardware, networks, and software to improve the performance and availability of a single computing Resource [17,18]. As a result of this integration, cluster computing systems provide users and applications with access to vast high performance computing resources by creating an illusion of a single system. Another common use of cluster computing is to provide load balancing on high-traffic websites [2]. Fig. 2 presents a cluster computing system that consists of:

- **Management servers:** responsible for controlling the system by taking care of system installation, monitoring, maintenance, and other management tasks.
- **Storage servers:** connected to disks and backup for the storage and data backup purposes respectively. The storage server also provides a shared file system access across the cluster.
- **User nodes:** used by system users to login to user nodes to run the workloads on each cluster.
- **Scheduler nodes:** used to distribute the users tasks on the computing nodes for processing.
- **Computer (Computing) nodes:** used to execute the workloads (user tasks assigned to it by the scheduler nodes).
2.1.1 Examples of CCS

Many examples of CCS have been built such as:

a. **ALICE**: ALICE stands for the Ames Lab-ISU Computing Environment. It consists of 64 dual-processor Pentium Pros running at 200 MHz. All PCs are connected through a central Fast Ethernet switch, providing a flat network topology. In addition, there is a master node, a file server, and 4 development nodes [22].

b. **G4 Cluster**: It consists of 16 single processor G4 computers each with 512 MB RAM and 16 dual processor G4s with 1 GB RAM. Each G4 is running Black Lab Linux and is connected via Fast Ethernet for management and Myrinet for communications [22].

c. **Sysplex**: The word sysplex comes from the first part of the word system and the last part of the word complex. It is a cluster-based approach for a mainframe system designed by IBM. It consists of multiple computers (the systems) that make up the complex. A sysplex is designed to be a solution for business needs involving any or all of the following: parallel processing; online transaction processing (OLTP); very high transaction volumes; very numerous small work units - online transactions, for example (or large work units that can be broken up into multiple small work units); or applications running simultaneously on separate systems that must be able to update to a single database without compromising data integrity [19, 23].

Other cluster computing systems have been built by leading companies, such as Microsoft, and Sun Microsystems offer clustering packages for scalability and availability [19].

Today's technology enables the extension of cluster class by incorporating load balancing, parallel processing, multi-level system management, and scalability methodologies [2]. Load balancing algorithms [4, 12-16, 20] are designed essentially to optimally distribute the system workload on computing resources aiming to maximize resources utilization, and that leads to minimizing the total task execution time. Since the communication cost represents a dominant factor in the task response time, the load-balancing strategy should be fair in distributing the system workload on the computing resources, and it should take into consideration the heterogeneity of these resources. The main objective of the load balancing strategies is to minimize the total communication and execution time encountered by the task assignment with respect to the resource constraints.

CCS have evolved to support applications ranging from e-commerce, to high performance database applications. Examples of these applications include Petroleum Reservoir Simulation, Protein Explorer, Earthquake Simulation, Image Rendering, and Whether Forecasting.

### 2.2 Grid Computing Systems (GCS)

GCS is a technology used to harness computing powers from various sources and use them in harmony to achieve a specific goal. The great advantage of GCS is the ability to significantly reduce the time that is taken to accomplish that goal, thereby increasing efficiency [26]. It utilizes the Internet as a medium for the wide spread availability of powerful computing resources as low-cost commodity components [2]. GCS can be thought of as a distributed system of logically coupled local clusters with non-interactive workloads that involve a large number of files. The word non-interactive means that assigned workload is treated as a single task. The logically coupled clustering refers that the output of one cluster may become input for another cluster, but within a cluster the workload is interactive. In contrast with the conventional cluster computing systems, grids account for different administrative domains with access policies, such as user privileges [24].

GCS supports the sharing and coordinated use of resources independently of their type and location in dynamic virtual organizations (VOs) consisting of individuals, institutions, and resources solving computationally intensive applications. It uses common interface to link computing clusters or LANs together. These clusters are shared between many users or VOs and a local policy is applied to each cluster that defines their rules for resource sharing. Such rules can be: what is shared on the basis of what condition and to whom, etc. Moreover, Grid guarantees the secure access by user identification [7]. What makes GCS different from CCS is that GCS tend to be more loosely coupled, heterogeneous, and geographically dispersed [24].

Most applications of grid computing are ones where the computing resources of one computing unit prove to be insufficient for the task at hand. The computing unit in question could potentially range from a single personal computer to a supercomputer within a large organization.

For example, a weather forecasting unit would require multiple variables and calculations within the program. Computing various scenarios and determining the probability of each scenario requires a large amount of computing power and time. The data that is required for such a task needs to be current, and the results need to be available within a certain time frame. This is an ideal application for grid computing. Fig.3. depicts an example of a Grid Computing model architecture.

---

**Fig.2: Cluster computing system architecture**

**Fig.3: Grid Computing model architecture**
2.2.1 Examples of GCS

Many projects for building GCS have been implemented. The following examples of GCS are listed in [26]:

a. **SETI Project**

   Is one of the most famous examples of Grid Computing (GC) projects. Search for Extra-Terrestrial Intelligence (SETI). The application looks for radio signals or other forms of communication in space, in an effort to prove the existence of extra-terrestrial intelligence.

   SETI developed a GC middleware where the process could be executed over multiple computers, since the application requires huge computing resources to scan the skies effectively. The infrastructure was designed in such a way that a layman using the Internet could choose to donate their unused computing power to the project.

b. **LHC Computing Grid**

   Another famous example of GC projects is the GC used to support the Large Hadron Collider at CERN (European Organization for Nuclear Research). The LHC GC is unlike the SETI@home project, in that it does not use donated processing power. It is a closed one, since only certain organizations have access to it.

c. **NFCR Centre for Computational Drug Discovery**

   It is a GCS at Oxford University Centre for Computational Drug Discovery. In this project, volunteers donate a few of the computing cycles when their screensavers are running. When screensavers are running, the computer is essentially idle, apart from a few background tasks. The project aims to find a cure for cancer using computational methods to screen small molecule structures, otherwise shortening a very lengthy process.

   There are many smaller GC projects that exist all over the world. Typically applications like weather forecasting, protein folding and earthquake simulation are prime candidates for a grid infrastructure. Grids have also been used to render large-scale animation projects, like movies.

2.3 **Cloud Computing (CC) Systems**

Cloud Computing (CC) is a large-scale distributed computing paradigm that is driven by economies of scale, in which a pool of abstracted, virtualized, dynamically-scalable, managed computing power, storage, platforms, and services are delivered on demand to external customers over the Internet [2,27-29]. Fig. 4. Presents a cloud computing model. The name CC was inspired by the cloud symbol that's often used to represent the Internet in flowcharts and diagrams. It describes a new model for Information Technology (IT) services based on the Internet. These services are accessible anywhere in the world, with the cloud appearing as a single point of access for all the computing needs of customers.

As illustrated in Fig. 5., cloud services are broadly divided into three categories [2,28,29]:

a. **Infrastructure-as-a-Service (IaaS):**

Cloud consumers directly use IT infrastructures (processing, storage, networks, and other fundamental computing resources) provided in the IaaS cloud. Virtualization is extensively used in IaaS cloud in order to integrate/decompose physical resources in an ad-hoc manner to meet growing or shrinking resource demand from cloud consumers. The basic strategy of virtualization is to set up independent virtual machines (VM) that are isolated from both the underlying hardware and other VMs. An example of IaaS is Amazon's EC2.

b. **Software-as-a-Service (SaaS).**

Cloud consumers release their applications on a hosting environment, which can be accessed through networks from various clients (e.g. web browser, PDA, etc.) by application users. Cloud consumers do not have control over the Cloud infrastructure that often employs a multi-tenancy system architecture, namely, different cloud consumers' applications are organized in a single logical environment on the SaaS cloud to achieve economies of scale and optimization in terms
of speed, security, availability, disaster recovery, and maintenance. Examples of SaaS include SalesForce.com, Google Mail, Google Docs, and so forth.

c. **Platform-as-a-Service (PaaS)**

PaaS is a development platform supporting the full "Software Lifecycle" which allows cloud consumers to develop cloud services and applications (e.g. SaaS) directly on the PaaS cloud. Hence the difference between SaaS and PaaS is that SaaS only hosts completed cloud applications whereas PaaS offers a development platform that hosts both completed and in-progress cloud applications. This requires PaaS, in addition to supporting application hosting environment, to possess development infrastructure including programming environment, tools, configuration management, and so forth. An example of PaaS is Google AppEngine.

These services sold on demand, typically by the minute or the hour; it is elastic -- a user can have as much or as little of a service as he wants at any given time; and the service is fully managed by the provider (the consumer needs nothing but a personal computer and Internet access). Significant innovations in virtualization and distributed computing, as well as improved access to high-speed Internet and a weak economy, have accelerated interest in cloud computing.

Clouds can be classified into three main types [28,29]:

a. **Public Cloud**

This is the dominant form of current Cloud computing deployment model where resources are dynamically provisioned on a fine-grained, self-service basis over the internet, via web applications/web services, from an offsite third-parity provider who share resources. A public cloud sells services to anyone on the Internet. (Currently, Amazon Web Services is the largest public cloud provider.)

b. **Private Cloud**

Data and processes are manipulated within the organization without any restrictions such as network bandwidth, security exposures, and legal requirements that using public cloud services might entail. Simply, one can say that a private cloud is a proprietary network or a data center that supplies hosted services to a limited number of people.

c. **Hybrid Cloud**

This cloud infrastructure is a combination of two or more clouds (private, or public) that remain unique entities but are bound together by standardized or proprietary technology that enables data and application portability. Organizations use the hybrid cloud model in order to optimize their resources to increase their core competencies by marging out peripheral business functions onto the cloud while controlling core activities on-premise through private cloud.

To be private, public, or hybrid, the goal of cloud computing is to provide easy, scalable access to computing resources and IT services. Fig. 6. Illustrates the cloud computing types.

2.3.1 **Examples of CC Systems**

The following are examples of cloud computing systems available today:

a. **Amazon Elastic Compute Cloud EC2**

It is a web service that provides resizable computing capacity in the cloud. It is designed to make web-scale computing easier for developers. Amazon EC2’s simple web service interface allows you to obtain and configure capacity with minimal friction. It provides users with complete control of their computing resources and lets them run on Amazon’s proven computing environment [30].

b. **Google App Engine**

Google App Engine applications are easy to build, easy to maintain, and easy to scale as users traffic and data storage needs change. With App Engine, there are no servers for you to maintain. You simply upload your application and it’s ready to go [31].

c. **Eucalyptus**

Eucalyptus is a free and open-source computer software for building Amazon Web Services (AWS)-compatible private and hybrid cloud computing environments marketed by the company Eucalyptus Systems. The Eucalyptus User Console provides an interface for users to self-service provision and configure compute, network, and storage resources that can be dynamically scaled up or down as application workloads change [32].

Some providers offer cloud computing services for free while others require a paid subscription.

3. **LOAD BALANCING IN DISTRIBUTED COMPUTING SYSTEMS**

Improving system performance is one of the most important issues in distributed computing systems. It is always required
to share work load among the various computing nodes of the distributed system to efficiently utilize available resources and as a direct result, the system performance is improved. Random distribution of tasks arriving to the computing nodes in distributed system may lead to a situation where some computing nodes become heavily loaded while others in the system may be lightly loaded or even idle. It is therefore desirable to transfer some tasks from the heavily loaded computers to the idle or lightly loaded ones aiming to efficiently utilize the available computing resources and hence improve the hole system performance. The process of load redistribution is known as load balancing [2-9]. This can help to avoid the situation where computing nodes are either heavily loaded or under loaded in the network.

In order to fulfill the user expectations in terms of performance and efficiency, a distributed computing system needs efficient load balancing algorithms for the distribution of the system's workload on the available resources. Load balancing in distributed computing systems environment has an important impact on the performance [2-9,13-18].

3.1 Benefits of Load Balancing

Balancing the system workload improves the performance of each computing node and hence the overall system performance is improved. The benefits of load balancing can be summarized as follows:

- Reduces the task waiting time
- Minimizes task response time
- Maximizes utilization of system resources
- Maximizes system throughput
- Improves reliability, and stability of the system.
- Accommodates future modification.

Because of the above benefits, the load balancing strategies become a field of intensive research [2-9, 13-18]. As a result, a large number of scheduling and load balancing algorithms have been developed in the past several years. Generally, there is no single load balancing algorithm that is appropriate for all applications and systems. The selection of an appropriate load balancing depends on application parameters like balancing quality, load generation patterns and also system parameters like resource heterogeneity, processing capacity, memory size, and communication bandwidth.

Generally load balancing algorithms/policies are classified into two types: Static Load balancing algorithms and Dynamic Load Balancing algorithms [2-9,13-18].

3.2 Static Load Balancing (SLB)

In the SLB algorithms, the load balancing decisions are made deterministically or probabilistically at compile time according to the performance of computing nodes and remain constant during runtime. Once the tasks are assigned, no change or reassignment is possible at the run time. Number of tasks in each node is fixed in SLB algorithms. SLB algorithms do not collect any information about the computing nodes [2-9, 13-18]. The assignment of tasks to the computing nodes is done based on many factors such as extent of resource needed, mean execution time arrival pattern, and inter-process communications. These factors should be measured before the assignment; this explains why the SLB algorithms are also called probabilistic algorithms.

As explained earlier, the load balancing decisions in SLB algorithms are not affected by the runtime system state information; instead they are affected only by the system statistical information. As a result, SLB algorithms have advantages in mathematical analysis and in implementation because of their simplicity. But estimating cost based on static information is not adaptive to situations such as one of the computing nodes selected to perform a computation fails, becomes isolated from the system due to network failure, or is so heavily loaded with tasks that its response time becomes longer than expected. Unfortunately, these situations are quite possible and beyond the capability of a traditional scheduler running static SLB algorithms [34]. As there is no migration of tasks at the runtime, the system overhead is minimized or even removed [4, 33]. A general disadvantage of SLB algorithms is that the final selection of a computing node for task allocation is made when the task is created and cannot be changed during process execution to make changes in the system load [2-9,13-18,33-37].

3.2.1 Optimal Static Load Balancing (OSLB)

If all the information and resources related to a system are known OSLB can be done. It is possible to increase throughput of a system and to maximize the use of the resources by using OSLB. For more information about OSLB, the reader is referred to [4,33]. Genetic algorithms (GA) and simulated annealing (SA) are examples of optimization techniques [9].

3.2.2 Sub-Optimal Static Load Balancing (S_OSLB)

S_OSLB algorithms will be mandatory for some applications and systems when optimal solution is not found. The thumb-rule and heuristics methods are important for sub-optimal algorithm [9].

3.2.3 Example of SLB algorithms

Many SLB algorithms have been proposed including Round Robin algorithm, Randomized algorithm, Central Manager Algorithm, and Threshold algorithm [4,38].

3.2.4 Round Robin Algorithm (RRA)

RRA assigns tasks sequentially to processing nodes. If number of tasks is greater than number processing nodes, the allocation wraps around. All tasks are assigned to computing nodes based on Round Robin order, meaning that computing nodes choosing is performed in series and will be back to the first computing node if the last computing node has been reached. The main advantage of RRA is that it does not require inter process communication. Generally, RRA does not achieve good performance because when the tasks are of unequal processing time, the RRA suffers as some of the computing nodes can become severely loaded while others remain idle. RRA is generally used in web servers where generally HTTP requests are of similar nature and thereby be distributed equally.
3.2.5 Randomized Algorithm (RA)

RA uses random numbers in selecting computing nodes for processing. The computing nodes are selected randomly following random numbers generated based on a statistic distribution [4-6,38].

3.2.5.1 Central Manager Algorithm (CMA)

In each step in the CMA, the master node (manager) will choose a computing node to be assigned a task. The chosen computing node is the one having the lightest workload. The master node is able to gather all computing nodes workload information. The load manager makes load balancing decisions based on the system load information, allowing the best decision when of the process created. High degree of inter-process communication could cause a bottleneck state to CMA.

3.2.5.2 Threshold Algorithm (TA)

In TA, tasks are assigned immediately upon creation to the computing nodes. Each computing node keeps a private copy of the system’s workload information. The load of a computing node can be characterized by one of the three levels: under loaded, medium and overloaded. Two threshold parameters \( t_{\text{under}} \) and \( t_{\text{upper}} \) can be used to describe these levels.

1. **Under loaded**: \( \text{workload} < t_{\text{under}} \),
2. **Medium**: \( t_{\text{under}} \leq \text{workload} \leq t_{\text{upper}} \),
3. **Overloaded**: \( \text{workload} > t_{\text{upper}} \).

Initially, all computing nodes are considered to be under loaded. When the load state of a computing node exceeds the load level limit, then it sends messages regarding the new load state to all of the other computing nodes, regularly updating them as to the actual load state of the entire system. If the local state is not overloaded then the task is allocated locally. Otherwise, a remote under loaded computing node is selected, and if no such computing nodes exists, the task is also allocated locally[4,38].

The SLB algorithms can be divided into two sub-classes: optimal static load balancing, and sub-optimal static load balancing [4,34].

3.3 Dynamic Load Balancing (DLB)

During the static load balancing, too much information about the system and tasks must be known before taking the load balancing decision. This information may not be available in advance. In contrast, DLB algorithms use the runtime system state information to make more informative load balancing decision, see [2-9, 14-19, 34-36] for more details. In DLB, Tasks assignment is done at the runtime. Also in DLB, tasks are reassigned at the runtime depending upon the situation that the load will be transferred from heavily loaded computing nodes to the lightly loaded ones [2-9, 14-19, 34-36]. In this case, the communication overheads occur and become more when the system workload increases. In DLB no decision is taken until the process gets execution stage. This strategy collects the information about the system state and about the task information. As more information is collected by an algorithm in a short time, potentially the algorithm can make better decision. DLB is mostly considered in heterogeneous system because it consists of nodes with different speeds, different communication bandwidths, different memory sizes, and variable external loads due to the system heterogeneity.

The advantage of DLB over the SLB is that the system need not be aware of run-time behavior of the applications before execution. It is particularly useful in a system where the primary performance goal is maximizing resource utilization, rather than minimizing runtime for individual jobs [37]. If a resource is assigned too many tasks, it may invoke a balancing policy to decide whether to transfer some tasks to other resources, and which tasks to transfer. According to who will initiate the balancing process, there are two different approaches: **sender-initiated** where a node that receives a new task but doesn’t want to run the task initiates the task transfer, and **receiver-initiated** where a node that is willing to receive a new task initiates the process [4,36,37].

3.3.1 Dynamic Load Balancing Issues

The following policies are considered during the design of a DLB algorithm [8,35]:

a. **Task Assignment Policy**: which assigns the tasks to the computing nodes according to the situation in a system.
b. **Load Calculation Policy**: which tells how to calculate the workload of a particular node in a system.
c. **Task Transfer Policy**: which determines whether a job is to be executed locally or remotely. This also defines when a node becomes overloaded.
d. **System State policy**: which tells whether a node is overloaded or lightly loaded.
e. **Priority Assignment Policy**: which tells the priority of execution of local and remote processes at a particular node.
f. **Information Exchange Policy**: which tells how to exchange the system work load information among the nodes. The information policy could use any of the following [4,33,35,36].

a. **On demand**: Scheduler begins to collect system workload information of system nodes when load balancing operation is going to start.
b. **Periodical**: Computing nodes send their workload information at regular time interval to the scheduler.
c. **On-state-change**: when the state of a computing node is changed, it immediately informs the others by passing its workload information.

DLB algorithms can be classified based on the methodology used in taking the load balancing decisions into two categories: Centralized, and distributed.
3.3.2 Centralized vs. Distributed

In DLB algorithms, the responsibility for making global scheduling decisions may lie with one centralized scheduler, or be shared by multiple distributed schedulers. In a computational distributed computing system, there might be many tasks submitted or required to be rescheduled simultaneously. 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 i.e., single point of failure [4-9,37].

3.3.3 Cooperative vs. Non-cooperative

Both of the SLB, and DLB algorithms can be cooperative or non-cooperative. If a distributed load balancing algorithm is adopted, the next issue that should be considered is whether the nodes involved in the task scheduling process are working cooperatively or independently (non-cooperatively). In the non-cooperative case, individual decision makers (schedulers) 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. In the cooperative case, each load balancing decision maker (scheduler) has the responsibility to carry out its own portion of the scheduling task, but all schedulers are working toward a common system-wide goal. Each system scheduler’s local policy is concerned with making decisions in concert with the other schedulers in order to achieve some global goal, instead of making decisions which will only affect local performance or the performance of a particular task [34-37].

3.3.4 Example of DLB algorithms

Because of its major effect on the performance of Distributed systems, DLB algorithms becomes an attractive research area form many researchers. As a direct result, a large number of DLB algorithms have proposed, designed, and implemented. Some of these algorithms are discussed below.

3.3.4.1 Random DLB Algorithm

In this algorithms, if a load imbalance is detected by the load balancing manager, the load manager selects randomly a computing node to receive some tasks from the heavily loaded ones. It does not check the workload state information of the selected computing node before sending the tasks for computing there. This algorithm neither maintains any local workload information nor sends any workload information to other computing nodes. Furthermore, it is simple to design and easy to implement. But it causes considerable communication overheads due to the random selection of lightly loaded computing nodes. Also as a result of the random selection of lightly loaded computing nodes, this algorithm may lead to imbalance cases. Hence, poor utilizing system resources consequently, the system performance may be degraded [5-9, 41].

3.3.4.2 Nearest Neighbor Algorithm

With nearest neighbor load balancing algorithms, the load balancing manager makes balancing decisions based on localized workload information and manages workload migrations within its neighborhood. The local load manager at each computing node takes the balancing decisions depending on the workload it has and the workload information to its immediate neighbors. By exchanging the load successively to the neighboring computing nodes the system attains a global balanced workload state. The nearest neighbor algorithm is mainly divided into two categories which are diffusion method and dimension exchange method. In the diffusion method a heavily or lightly loaded computing node balances its workload simultaneously with all its nearest neighbors at a time while in dimension exchange method a computing node balances its load successively with its neighbor one at a time [9, 38-43].

3.3.4.3 Probabilistic algorithm

In this algorithm, each computing node keeps a load vector including the load of a subset of computing nodes. The first half of the load vector holds also the local workload which is sent periodically to a randomly selected node. The system workload information is updated in this way to minimize the communication overhead. This algorithm is not scalable i.e., its extensibility is poor [9].

3.3.4.4 Threshold Algorithms

Theses algorithms use a partial knowledge about system workload information obtained by message exchanges. A computing node is randomly selected for accepting the migrated tasks. If the selected node workload is below a certain threshold limit, the migrated tasks are accepted for computing there. Otherwise, polling is repeated with another computing node for finding appropriate one [9,33].

3.3.4.5 Centralized Information and Centralized Decision

In these algorithms, the whole system workload information is stored at a single node (master node) and the load balancing decisions are also taken by that node. Every computing node in the system updates the master node by its workload information either periodically or upon the occurrence of any change in its workload. If a load imbalance is detected, the heavily loaded computing node requests the master node for a lightly loaded node to forward some of its workload to it. This kind of DLB algorithms suffer from a very serious problem that if the master node is crashed or not functioning well, the whole process will be stopped i.e., single point of failure. As a direct result of the single point of failure problem, also, these kind of DLB algorithms are not scalable [9].

3.3.4.6 Centralized Information and Distributed Decision

In these algorithms, collection of system workload information is centralized (at a master node) while decision making is distributed [9]. The system workload information of the computing nodes is broadcasted by the master node. Using
this information, an overloaded computing node finds the lightly loaded ones. This algorithm is very efficient due to the less inclusion of message information and it is robust in nature because the system remains alive even when the master node is in recovery state. This class of DLB algorithms collects large amount of system workload information but it may be not up-to-date. As a result greater overheads occur in the system [9,33].

3.3.4.7 Distributed Information and Distributed Decision

In this kind of DLB algorithms, each computing node updates the other computing nodes by its workload state information using any of the following polices:

- **Periodic broadcast**: each computing node broadcasts its workload state information only when the state of the node changes because of a task arrival or departure.
- **On demand Exchange**: A computing node broadcasts a state information request message when its state switches from normal to under loaded or overloaded region. Then the other computing nodes send their current workload state information.

Each computing node keeps a global workload information vector based on which, it takes the load balancing decisions. In case of periodic update, the length of the time period for workload information update has a great impact on the system performance. Because, if it is too short, this will make the load balancing decision to be taken based on up-to-date workload information. But on the other hand, this will increase the system communication overhead which may degrade the performance of this kind of DLB algorithms [4-9]

3.4 Comparison Between SLB and DLB Algorithms

DLB algorithms take the load balancing decision based on the current state of the system workload information while SLB algorithms take the load balancing decisions based strictly on a fixed and pre-configured set of rules relating to characteristics of the input traffic. For this reason it is natural to thought that DLB algorithms outperform the static ones. But this is not always the case; it has been proved that SLB algorithms outperform the DLB ones [4,8,32,44]. This section presents the main performance indicators that can be used to compare between any SLB and DLB algorithms.

3.4.1 Nature

This factor is related with determining the nature or behavior of load balancing algorithms that is whether the algorithm is of static or dynamic nature, pre-planned or no planning. SLB algorithms are of static and planned nature as tasks are assigned statically i.e. at compile time in a planned manner to computing nodes and there will be no redistribution of tasks takes place afterwards and outcome of the algorithm is deterministic as much of the task information is known a priori. In contrast, DLB algorithms are of dynamic and no planning nature as tasks are assigned at runtime to processors and tasks redistribution can take place if task assignment that was earlier done is not giving good performance (that is if proper balancing of load is not there). So their behavior is totally nondeterministic and no initial planning is done for assigning load to hosts as this work is done at run-time [4,8,32,44].

3.4.2 Associated Overhead

This performance indicator is concerned by determining the amount of overhead involved while implementing a load-balancing algorithm. It is composed of overhead due to movement (relocation) of tasks, inter-processor communication, and inter-process communication. In SLB algorithms, once tasks are assigned to computing nodes, no redistribution of tasks takes place, so no relocation overhead. Also, SLB algorithms do not collect current system state information, so no inter-processor communication overhead. But a little overhead may occur due to the inter process communications. In contrast, as the DLB algorithms takes the load balancing decisions based on the current system state information, and it distributes tasks at the run time, it may suffer from all these types of overheads, especially at heavy workloads. As a result, one can say that SLB algorithms involve a less amount of overheads than the static ones [4,8,32,44].

3.4.3 Resource Utilization

This performance indicator is concerned with determining the efficiency of utilizing available system resources while implementing a load-balancing algorithm. SLB algorithms have poor resource utilization capability because it just tries to assign tasks to the computing nodes in order to minimize response time ignoring the fact that may be using this task assignment can result into a situation in which some computing nodes finish their work early and sit idle due to the absence of a task reassignment policy. DLB algorithms have relatively better resource utilization as it takes care of the fact that load should be evenly distributed to computing nodes so that no node can sit idle while others work. This because in it, tasks are reassigned at the runtime depending upon the situation that the workload will be transferred from heavily loaded computing nodes to the lightly loaded one i.e., it has a task reassignment policy. Generally, one can say that DLB algorithms can utilize system resources much better than static ones [4,8,32,44].

3.4.4 Preemptiveness

This performance indicator is concerned by checking the fact that whether tasks in execution can be transferred to other computing nodes (processors) or not. SLB algorithms are inherently non-preemptive as no tasks are relocated. DLB algorithms are both preemptive and non preemptive [4,8,32,44].

3.4.5 Predictability

This performance indicator is related with the deterministic or nondeterministic factor that is to predict the outcome of the algorithm. SLB algorithm’s behavior is predictable as most of the things like average waiting, execution, and response time of tasks and workload assignment to computing nodes are
fixed at compile-time. DLB algorithm’s behavior is unpredictable, as everything has been done at run time [4,8,32,44].

3.4.6 Adaptable

This performance indicator is used to check whether the algorithm is adaptive to varying or changing situations i.e. situations which are of dynamic nature. SLB algorithms are not adaptive towards all circumstances as this method fails in dynamic or varying nature problems i.e. situation in which number of processes are not fixed, also in situations which may require indeterminate steps towards solution. DLB algorithms are adaptive towards every situation whether numbers of processes are fixed or varying one [4,8,32,44].

3.4.7 Reliability

This performance indicator is used to check which load balancing algorithm is more reliable in case of some computing nodes failure occurrence. SLB algorithms are less reliable because no task will be relocated / transferred to another computing node in case a node fails at run-time. DLB algorithms are relatively more reliable as in it, tasks can be transferred to other computing nodes in the case of node failure occurrence [4,8,32,44].

3.4.8 Stability

This performance indicator is related to the exchange of present workload state information among computing nodes. SLB algorithms in this context can be considered as stable as no information regarding present workload state is passed among computing nodes. However in case of DLB such kind of information is exchanged among computing nodes and if this information is out of date i.e. information which is not updated regularly or periodically among computing nodes then it can lead the whole system to an unstable state [4,8,32,44].

4. CONCLUSION AND FUTURE WORK

As a direct result for the rapid development of computing resources, the performance of computers has enhanced and their cost is reduced. This availability of low cost powerful computers coupled with the advances and popularity of the Internet and high speed networks led the computing environment to be mapped from the traditionally distributed systems to clusters, grid, and cloud computing environments. These newly emerged computing environments are called HPDCS. One of the main motivations of these systems is to aggregate the power of widely distributed resources, and provide non-trivial services to users. To achieve this goal, efficiently utilizing system resource is an important research issue in HPDCS. Utilizing available system resources can be done by using a good load balancing algorithm. This survey paper provides a review of the subject mainly from the perspective of HPDCS, and their load balancing algorithms. In this review, the challenges of the HPDCS and their load balancing algorithms are identified. First, the HPDCS are classified into three broad categories, namely: (a) cluster, (b) grid, and (c) cloud systems, and are briefly introduced to provide an intuitive image of these systems and their important role in our recent life. Then a comprehensive discussion of widely used load balancing algorithms deployed in HPCS environment is presented. Finally, a classification of these algorithms is done from different points of view, such as static vs. dynamic, centralized vs. decentralized, and finally cooperative vs. non-cooperative algorithms.

In the future, we will try to conduct a comparative study on the effect of various load balancing algorithms in improving performance of HPDCS.

REFERENCES

March 2004, he was a Ph. D. candidate at the Institute of Information Sciences and Electronics, University of Tsukuba, Japan, where he was conducting research on aspects of load balancing in distributed and parallel computer systems. From April 2004 to 2007, he worked as a lecturer of computer science, Faculty of Science, Menoufia University, Egypt. Currently, he is working as an assistant professor of computer science at College of Computing and Information Technology, Taif University, Kingdom of Saudi Arabia. His research interests are performance evaluation, load balancing in distributed/parallel computing systems, Grid computing, network security and cryptography.

Prof. Dr. Said Ghoniemy received his B.Sc. degree in Electrical Engineering, and his M.Sc. degree, in Communication Engineering, from Military Technical College (MTC) Cairo-Egypt in 1968 and 1979 respectively. He received his Ph. D. from “l’Institut National Polytechnique de Toulouse (INP)”, France in 1982. He is currently a Professor in the Department of Computer Systems, Faculty of computers and Information Sciences, Ain Shams University, Cairo, Egypt, and now he is a Professor of Computer Engineering at Taif University, KSA. He served as a director of the Information Technology Research & Consultancy Center (ITRCC), Ain Shams University, for more than 6 years. He has been the Vice Dean of the Faculty of computers and Information Sciences, Ain Shams University. His research interests include Computer Systems, Signal Processing, Robotics & Computer Vision, and Mobile Ad-hoc Networks.

Dr. Said El-Zoghdy received his BSc degree in pure Mathematics and Computer Sciences in 1993, and MSc degree for his work in computer science in 1997, all from the Faculty of Science, Menoufia, Shebin El-Koom, Egypt. In 2004, he received his Ph. D. in Computer Science from the Institute of Information Sciences and Electronics, University of Tsukuba, Japan. From 1994 to 1997, he was a demonstrator of computer science at the Faculty of Science, Menoufia University, Egypt. From December 1997 to March 2000, he was an assistant lecturer of computer science at the same place. From April 2000 to