Parallel Computing 39 (2013) 709–736

Contents lists available at ScienceDirect

Parallel Computing

journal homepage: www.elsevier.com/locate/parco

Review

A survey on resource allocation in high performance distributed computing systems

Hameed Hussain [a], Saif Ur Rehman Malik [b], Abdul Hameed [b], Samee Ullah Khan [b,*], Gage Bickler [b], Nasro Min-Allah [a], Muhammad Bilal Qureshi [a], Limin Zhang [b], Wang Yongji [c], Nasir Ghani [d], Joanna Kolodziej [e], Albert Y. Zomaya [f], Cheng-Zhong Xu [g], Pavan Balaji [h], Abhinav Vishnu [i], Fredric Pinel [j], Johnatan E. Pecero [j], Dzmitry Kliazovich [j], Pascal Bouvry [j], Hongxiang Li [k], Lizhe Wang [l], Dan Chen [m], Ammar Rayes [n]

[a] COMSATS Institute of Information Technology, Islamabad 44000, Pakistan
[b] North Dakota State University, Fargo, ND, USA
[c] Institute of Software, Chinese Academy of Sciences, Beijing, China
[d] University of South Florida, Tampa, Florida 33620-5399, USA
[e] Cracow University of Technology, Cracow, Poland
[f] University of Sydney, Sydney, NSW, Australia
[g] Wayne State University, Detroit, MI, USA
[h] Argonne National Laboratory, Argonne, IL, USA
[i] Pacific Northwest National Laboratory, Richland, WA, USA
[j] University of Luxembourg, Coudenhove-Kalergi, L1359, Luxembourg
[k] University of Louisville, Louisville, KY, USA
[l] Center for Earth Observation and Digital Earth, Chinese Academy of Sciences, Beijing, China
[m] China University of Geosciences, Wuhan, China
[n] CISCO Systems, San Jose, CA, USA

ARTICLE INFO

Article history:
Received 24 September 2011
Received in revised form 10 September 2013
Accepted 16 September 2013
Available online 14 October 2013

Keywords:
Scheduling
Resource allocation
Resource management

ABSTRACT

An efficient resource allocation is a fundamental requirement in high performance computing (HPC) systems. Many projects are dedicated to large-scale distributed computing systems that have designed and developed resource allocation mechanisms with a variety of architectures and services. In our study, through analysis, a comprehensive survey for describing resource allocation in various HPCs is reported. The aim of the work is to aggregate under a joint framework, the existing solutions for HPC to provide a thorough analysis and characteristics of the resource management and allocation strategies. Resource allocation mechanisms and strategies play a vital role towards the performance improvement of all the HPCs classifications. Therefore, a comprehensive discussion of widely used resource allocation strategies deployed in HPC environment is required, which is one of the motivations of this survey. Moreover, we have classified the HPC systems into three broad categories, namely: (a) cluster, (b) grid, and (c) cloud systems and define the characteristics of each class by extracting sets of common attributes. All of the

* Corresponding author. Address: Department of Electrical and Computer Engineering, North Dakota State University, Fargo, ND 58108-6050, USA. Tel.: +1 701 231 7615; fax: +1 701 231 8677.

E-mail addresses: ham.hamdard@gmail.com (H. Hussain), saif.rehmanmalik@ndsu.edu (S.U.R. Malik), abdul.hameed@ndsu.edu (A. Hameed), samee.khan@ndsu.edu (S.U. Khan), gage.n.bickler@ndsu.edu (G. Bickler), nasar@comsats.edu.pk (N. Min-Allah), muhdbilal.qureshi@gmail.com (M.B. Qureshi), limin.zhang@ndsu.edu (L. Zhang), ywang@itechs.iscas.ac.cn (W. Yongji), nghani@usf.edu (N. Ghani), jkolodziej@uck.pk.edu.pl (J. Kolodziej), albert.zomaya@sydney.edu.au (A.Y. Zomaya), czxu@wayne.edu (C.-Z. Xu), balaji@mcs.anl.gov (P. Balaji), abhinav.vishnu@pnl.gov (A. Vishnu), fredric.pinel@uni.lu (F. Pinel), johnatan.pecero@uni.lu (J.E. Pecero), dzmitry.kliazovich@uni.lu (D. Kliazovich), pascal.bouvry@uni.lu (P. Bouvry), h.li@louisville.edu (H. Li), lzwang@ceode.ac.cn (L. Wang), chendan@pmail.ntu.edu.sg (D. Chen), rayes@cisco.com (A. Rayes).

http://dx.doi.org/10.1016/j.parco.2013.09.009

aforementioned systems are cataloged into pure software and hybrid/hardware solutions. The system classification is used to identify approaches followed by the implementation of existing resource allocation strategies that are widely presented in the literature.

Contents

1. Introduction

The distributed computing paradigm endeavors to tie together the power of large number of resources distributed across a network. Each user has the requirements that are shared in the network architecture through a proper communication channel [1]. Distributed computing paradigm is used for three major reasons. First, the nature of 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. Second, most of the parallel applications have multiple processes that run concurrently on many nodes communicating over a high-speed interconnect. The use of high performance distributed systems for parallel applications is beneficial as compared to a single Central Processing Unit (CPU) machine for 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 [2]. Third, 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. Some techniques for achieving reliability on a distributed environment are check pointing and replication [2]. Scalability, reliability, information sharing, and information exchange from remote sources are the main motivations for the users of distributed systems [2].

The resource management mechanism determines the efficiency of the used resources and guarantees the Quality of Service (QoS) provided to the users. Therefore, the resource allocation mechanisms are considered a central theme in HPCs [85]. Some applications require strict delay, computational power, and best effort services. Moreover, if the desired performance is not achieved, then the users feel reluctant to pay. Therefore, a growing need for QoS resource management and scheduling algorithms is observed [85]. QoS resource management aims at providing guaranteed deterministic services to Service Level Agreement (SLA) [39] based premium users and fair services to the best users. The users that do not require performance bounds are known as best users [85]. QoS resource management and scheduling algorithms are capable of optimally assigning resources in ideal situation or near-optimally assigning resources in actual situation, taking into account the task characteristics and QoS requirements [77,86].

Scheduling is the method by which system resources are allocated to the jobs, such as processor time and bandwidth, to balance the load effectively and to achieve a targeted QoS [137]. The need for a scheduling algorithm arises from the requirement for most modern systems to perform multitasking (execute more than one process at a time) and multiplexing (transmit multiple flows simultaneously). Therefore, we must consider job scheduling as a resource management technique.

1.1. Motivation

The purpose of this survey is to analyze the resource allocation mechanism of three broad classes of HPC: (a) cluster, (b) grid, and (c) cloud. Besides other factors, the performances of the aforementioned classes are directly related to the resource allocation mechanisms used in the system. Therefore, in the said perspective, a complete analysis of resource allocation mechanism used in HPCs classes is required. In this survey, we present a thorough analysis and characteristics of the resource management and allocation strategies used in academic, industrial, and commercial system.

The features of the HPC categories (cluster, grid, and cloud) are conceptually similar [112]. Therefore, an effort has been made to distinguish each of the categories by selecting relevant distinct features for all. The features are selected based on the information present in the resource allocation domain, acquired from a plethora of literature. We believe that the comprehensive analysis of leading research and commercial projects in HPC domain can provide readers with an understanding of the essential concepts of the evolution of the resource allocation mechanisms in HPC systems. Moreover, this research will help individuals and researchers to identify the important and outstanding issues for further investigation. The highlighted aspects of the survey are as follows:

(a) Analysis of resource allocation mechanisms of cluster, grid, and cloud.
(b) Identifying the common features of each category and comparing the resource allocation mechanisms of the systems based on the selected features.
(c) Classification of systems as software only and hybrid/hardware systems.

In contrast to the other compact surveys and system taxonomies, such as [101,102], the focus of this study is to demonstrate the resource allocation mechanisms. Note that the purpose of this survey is to demonstrate the resource allocation mechanisms and not the performance analysis of the systems. Although, the performance can be analyzed based on the resource allocation mechanism but this is not the scope of the paper. The purpose of this study is to aggregate and analyze the existing solutions for HPC under the resource allocation policies. Moreover, an effort has been made to provide a broader view of the resource allocation mechanisms and strategies by discussing systems of different categories, such as obsolete systems (systems that were previously being used), academic system (research projects proposed by institutes and universities), and established systems (well-known working systems). The projects are compared on the basis of the selected common features within the same category. For each category, the characteristics discussed are specific and the list of features can be expanded further. Finally, the systems are cataloged into pure software and hybrid/hardware HPC solutions.

The rest of the paper is organized as follows: In Section 2, we present the HPC system classification and highlight the key terms and the basic characteristics of each class. In Section 3, we survey the existed HPC system research projects and commercial approaches of each classification (cluster, grid, and cloud). The projects are cataloged into pure software and hybrid/hardware solutions in Section 4. The paper concludes in Section 5 with some final remarks and open issues.

2. Overview of HPC systems

The section discusses three main categories of HPC systems that are analyzed, evaluated, and compared based on the set of identified features. We put cloud under the category of HPC because it is now possible to deploy a HPC cloud, such as Amazon EC2. Clusters having 50,000 cores have been run on Amazon EC2 for scientific applications [123]. Moreover, the HPC workload is usually massively high scale and has to be run on many machines, which is naturally compatible with a cloud environment. The taxonomy representing the categories and the selected features used for the comparison within the same category are shown in Fig. 1. Dong et al. [122] designed a taxonomy for the classification of scheduling algorithms in distributed systems. Moreover, Ref. [122] has broadly categorized scheduling algorithms as: (a) Local vs. Global, (b) Static vs. Dynamic, (c) Optimal vs. Suboptimal, (d) Distributed vs. Centralized, and (e) Application centric vs. Resource centric. Apart from above classification, different variants of scheduling, such as conservative, aggressive, and no reservation can also be found in literature [17,100]. In conservative scheduling [87], processes allocate required resources before execution. Moreover, the operations are delayed for serial execution of the tasks that helps in process sequencing. The delay is also helpful in rejection of the processes. In conservative scheduling, operations are not rejected but delayed. In an aggressive (easy) scheduling [87], operations are immediately scheduled for execution to avoid delay in the operations. Moreover, the operations are reordered on the arrival of new operations. In some situations when a task cannot be completed in serial way, the operations are rejected. In aggressive scheduling, the operations are not delayed but have rejection risk at later stages. However, in conservative scheduling the operations are not rejected but delayed. No reservation [88] is a dynamic scheduling technique where

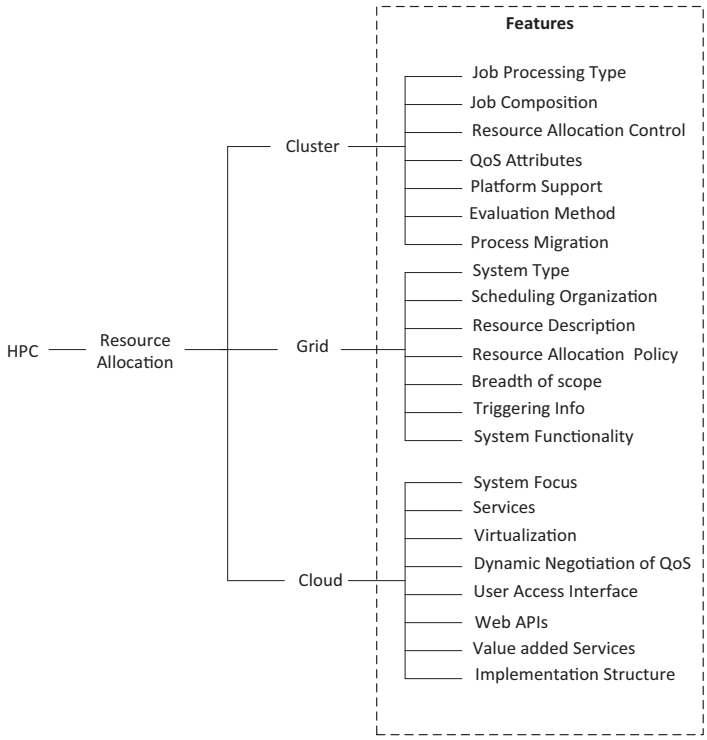


Fig. 1. HPC systems categories and attributes.

the resources are not reserved prior to execution but allocated at run time. The resources without reservation are wasted because, if a resource is not available at request time, then the process has to wait till the availability of resource.

2.1. HPC systems classes

2.1.1. Cluster computing systems

Cluster computing, referred as clustering, is the use of multiple computers, multiple storage devices, and redundant interconnections to form a single highly available system [143]. Cluster computing can be used for high availability and load balancing. A common use of cluster computing is to provide load balancing on high-traffic websites. The concept of clustering was already present in DEC's VMS systems [106,107]. IBM's Sysplex is a cluster-based approach for a mainframe system

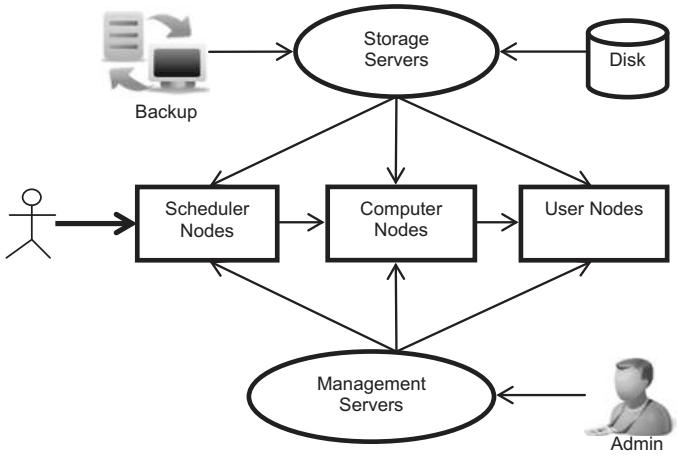


Fig. 2. A cluster computing system architecture.

[108]. Microsoft, Sun Microsystems, and other leading hardware and software companies offer clustering packages for scalability and availability [109]. With the increase in traffic or availability assurance, all or some parts of the cluster can be increased in size or number.

The goal of cluster computing is to design an efficient computing platform that uses a group of commodity computer resources integrated through hardware, networks, and software to improve the performance and availability of a single computer resource [144,145]. One of the main ideas of cluster computing is to portray a single system image to the outside world. Initially, cluster computing and HPC were referred to the same type of computing systems. However, today's technology enables the extension of cluster class by incorporating load balancing, parallel processing, multi-level system management, and scalability methodologies. Load balancing algorithms [104] are designed essentially to equally spread the load on processors and maximize the utilization while minimizing the total task execution time. To achieve the goals of cluster computing, the load-balancing mechanism should be fair in distributing the load across the processors [146]. The objective is to minimize the total execution and communication cost encountered by the task assignment, subject to the resource constraints.

The extension of traditional clusters transforms into user-demand systems (provides SLA-based performance) that deliver Reliability, Availability, and Serviceability (RAS) needed for HPC applications. A modern cluster is made up of a set of commodity computers that are usually restricted to a single switch or group of interconnected switches within a single virtual local-area network (VLAN) [93]. Each compute node (computer) 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. Fig. 2 depicts a cluster computing system that consists of: (a) Management servers (responsible for controlling the system by taking care of system installation, monitoring, maintenance, and other tasks), (b) Storage servers, disks, and backup (storage servers are connected to disks for the storgae purpose and the disks are connected to backup for data backup purposes, the storgae server in Fig. 2 provides a shared file system access across the cluster), (c) User nodes (used by system users to login to user nodes to run the workloads on each cluster), (d) Scheduler nodes (users submit their work to a scheduler nodes to run the workload), and (e) Computer nodes (run the workloads).

2.1.2. Grid computing systems

The concept of grid computing is based on using the Internet as a medium for the wide spread availability of powerful computing resources as low-cost commodity components [142]. Computational grid can be thought of as a distributed system of logically coupled local clusters with non-interactive workloads that involve a large number of files [15,111]. By non-interactive, we mean 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 HPC (cluster) systems, grids account for different administrative domains with access policies, such as user privileges [94,141]. Fig. 3 depicts a general model of grid computing system.

The motivations behind grid computing were the resource sharing and problem solving in multi-institutional and dynamic virtual organizations as depicted in Fig. 3. A group of individuals and institutions form a virtual organization. In virtual organization, the individuals and the institutions define rules for resource sharing. Such rules can be: what is shared on the

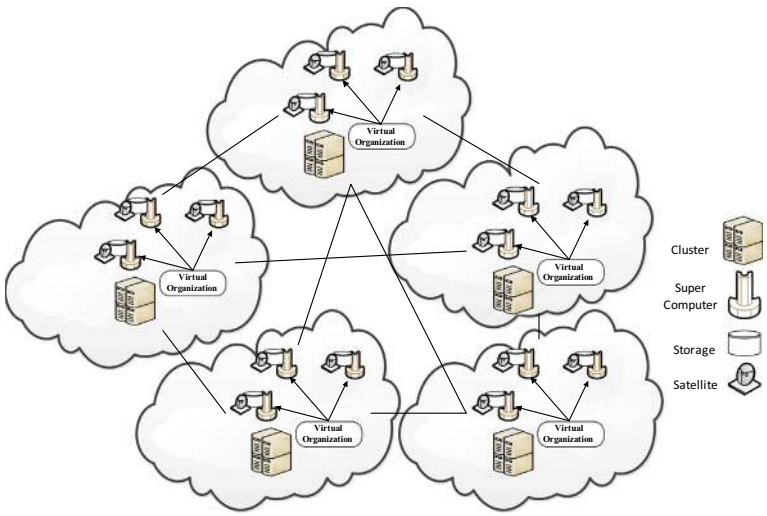


Fig. 3. A model of grid computing system.

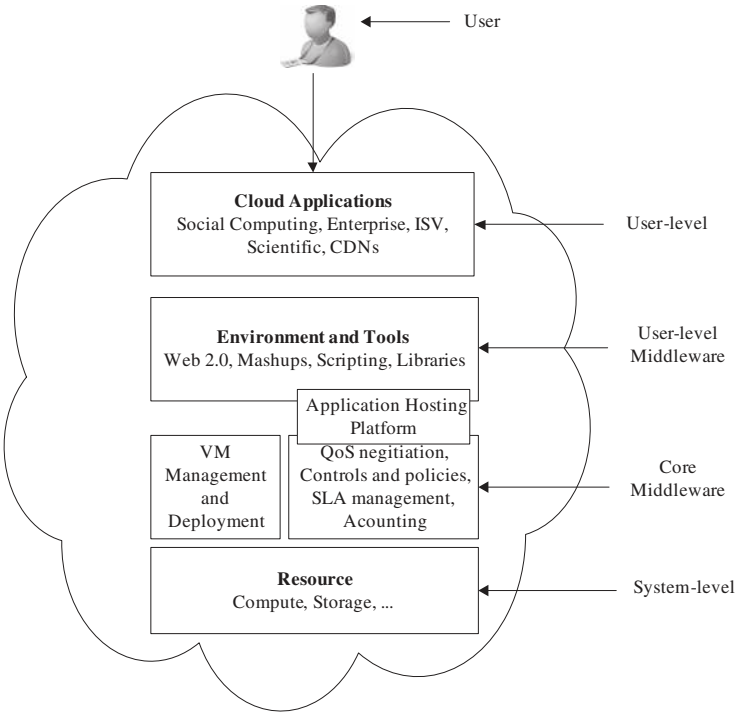


Fig. 4. A layered model of cloud computing system.

basis of what condition and to whom, etc. [95]. Moreover, Grid guarantees the secure access by user identification. The aggregate throughput is more important than the price and overall performance of a grid system. What makes grid different from conventional HPC systems, such as cluster, is that grids tend to be more loosely coupled, heterogeneous, and geographically dispersed [139,140].

2.1.3. Cloud computing systems

Cloud computing describes a new model for Information Technology (IT) services based on the Internet, and typically involves provision of dynamically scalable and often virtualized resources over-the-Internet [2,3,12,147]. Moreover, cloud computing provides the ease-of-access to remote computing sites using the Internet [4,5]. Fig. 4 shows a generic layered model of cloud computing system. The user-level layer in Fig. 4 is used by the users to deal with the service provided by the cloud. Moreover, the top layer also uses the services provided by the lower layer to deliver the capabilities of SaaS [120]. The tools and environment that are required to create interfaces and applications on the cloud is provided by the user-level middleware layer. The runtime environment that enables cloud computing capabilities to application services of user-level middleware is provided by the Core middleware layer. Moreover, the computing capabilities are provided by the layer through implementing the platform level services [120]. The computing and processing power of cloud computing is aggregated through data centers [148–150]. At the system level layer physical resources, such as storage servers and application servers are available that powers up the data center [120].

The current cloud systems, such as Amazon EC2 [68], Eucalyptus [71], and LEAD [121] are based mainly on the Virtual Grid Application Development Software (VGrADS), sponsored by National Institute of Standards and Technology (NIST) [6]. The term "Cloud" is a metaphor for the Internet. The metaphor is based on the cloud drawing used in the past to represent the telephone network [7] and later to depict the Internet in computer network diagrams as an abstraction of the underlying infrastructure [8]. Typical cloud computing providers deliver common business applications online that are accessed through web service, and the data and software are stored on the servers. Clouds often appear as a single point of access for computing the consumer needs. Commercial offerings are generally expected to meet the QoS requirements of customers, and typically include SLAs [9].

The model of the cloud requires minimal management and interactions with IT administrators and resource providers, as seen by the user. Alternatively, self-monitoring and healing of cloud computing system requires complex networking, storage, and intelligent system configuration. Self-monitoring is necessary for automatic balancing of workloads across the physical network nodes to optimize the cost of system utilization. Failure of any individual physical software or hardware component of the cloud system is arbitrated swiftly for rapid system recovery.

Table 1 depicts the common attributes among the HPC categories, such as size, network type, and coupling. Moreover, no numeric data is involved in the Table 1. For example, the size of the grid is large as compared to cluster. The network grids

Table 1
Commonality between cluster, grid, and cloud systems.

Feature	Cluster	Grid	Cloud
Size	Small to medium	Large	Small to large
Network type	Private, LAN	Private, WAN	Public, WAN
Job management and scheduling	Centralized	Decentralized	Both
Coupling	Tight	Loose/tight	Loose
Resource reservation	Pre-reserved	Pre-reserved	On-demand
SLA constraint	Strict	High	High
Resource support	Homogeneous and heterogeneous (GPU)	Heterogeneous	Heterogeneous
Virtualization	Semi-virtualized	Semi-virtualized	Completely virtualized
Security type	Medium	High	Low
SOA and heterogeneity support	Not supported	Supported	Supported
User interface	Single system image	Diverse and dynamic	Single system image
Initial infrastructure cost	Very high	High	Low
Self service and elasticity	No	No	Yes
Administrative domain	Single	Multi	Both

are usually private and over Wide Area Network (WAN) that means that grids spread over the Internet are owned by a single company. Foster et al. [101] uses various perspectives, such as architecture, security model, business model, programming model, virtualization, data model, and compute model to compare grids and clouds. Sadashiv et al. [102] have done a comparison of three computing models (cluster, grid, and cloud) based on different characteristics, such as business model, SLA, virtualization, and Reliability. Similar comparison can also be found in [98]. Another comparison amongst the three computing models can also be found [103].

2.2. Computer clusters: features and requirements

The overall performance of the cluster computing system depends on the features of the system. Cluster systems provide a mature solution for different types of computation and data-intensive parallel application. Among many specific system settings related to a particular problem, sets of basic generic cluster properties can be extracted as a common class of classical and modern cluster systems [118]. The extracted features shown in Fig. 1 are defined in the following paragraphs.

2.2.1. Job processing type

Jobs submitted to the cluster system may be processed as parallel or sequential. The jobs can be characterized as sequential or parallel based on the processing of the tasks involved in the job. A job that consists of parallel tasks has to execute concurrently on different processors, where each task starts at the same time. (The readers are encouraged to see [130] for more details on HPC job scheduling in Cluster.) Usually, the sequential jobs are executed at a single processor as a queue of independent tasks. Parallel applications are mapped to the multi-processor parallel machine and are executed simultaneously on the processors. The parallel processing mode speeds up the whole job execution and the appropriate strategy is to solve the complex large-scale problems within a reasonable amount of time and cost. Many conventional market-based cluster resource management systems support sequential processing mode. However, number of compute intensive applications must be executed within a feasible deadline. Therefore, parallel processing mode of cluster job is implemented in high-level cluster systems, such as SLURM [91], Enhanced MOSIX [2], and REXEC [3].

2.2.2. QoS attributes

QoS attributes describe the basic service requirements requested by the consumers that the service provider is required to deliver. The consumer represents a business user that generates service requests at a given rate that is to be processed by the system. General attributes involved in QoS are: (a) time, (b) cost, (c) efficiency, (d) reliability, (e) fairness, (f) throughput, (g) availability, (h) maintainability, and (i) security. QoS metrics can be estimated by using various measurement techniques. However, such techniques are difficult to use in solving a resource allocation problem with multiple constraints. The difficulty in resource allocation problem with multiple constraints is still a critical problem in cluster computing. In some conventional cluster systems, REXEC [3], Cluster-on-Demand [4], and LibraSLA [5], the job deadline and user defined budget constraints, such as: (a) fairness, (b) time, and (c) cost QoS attributes are considered. Market-based cluster RMS still lacks efficient support of reliability or trust. Recent applications that manipulate huge bytes of distributed data must provide guaranteed QoS during network accessibility. Providing the best effort services by ignoring the network mechanism is not enough to the customer requirements. (Readers are encouraged to read [131,132] for more understanding of QoS attribute in clusters.)

2.2.3. Job composition

Job composition depicts the number of tasks involved in a single job prescribed by the user. A single-task job is defined as a monolithic application, in which just a single task is specified as depicted in Fig. 5(a). Parallel (or multi-task) jobs are usu-

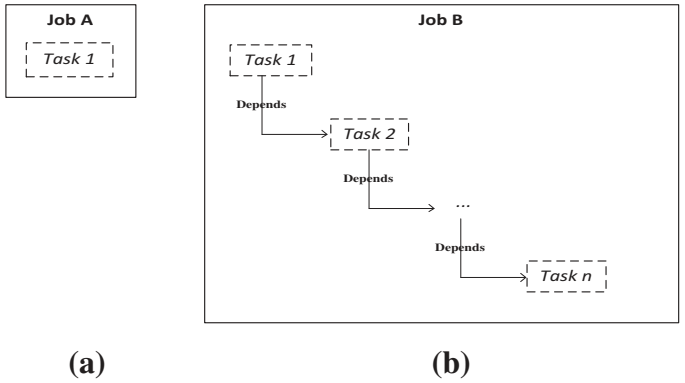


Fig. 5. (a) Single task job and (b) Multiple task job.

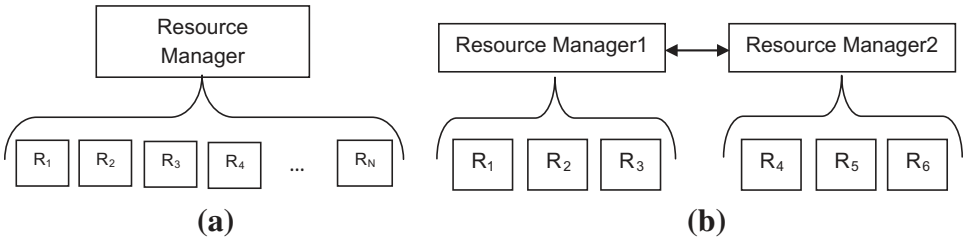


Fig. 6. Resource management (a) Centralized resource management (b) Decentralized resource management.

ally represented by a Directed Acyclic Graph (DAG) as shown in Fig. 5(b). Moreover, the nodes express the particular tasks partitioned from an application and the edges represent the inter-task communication [96]. The tasks can be dependent or independent. Independent tasks can be executed simultaneously to minimize the processing time. Dependent tasks are cumbersome and must be processed in a pre-defined manner to ensure that all dependencies are satisfied. Market-based cluster RMSs must support all three types of job compositions namely: (a) single task, (b) independent multiple-task, and (c) dependent multiple-task [96].

2.2.4. Resource allocation control

Resource allocation control is a mechanism that manages and control resources in a cluster system. Resource allocation control system can be centralized or decentralized [98]. The jobs in centralized system are being administered centrally by a single resource manager that has complete knowledge of the system. In decentralized resource management system (RMS), several resource managers and providers communicate with one another to keep the load for all resources balanced and satisfy the specific users requirements [98]. Fig. 6 depicts centralized and decentralized resource management systems (more details please see [133]).

2.2.5. Platform support

Two main categories of cluster infrastructure to support the execution of cluster applications are homogeneous and heterogeneous platforms. In a homogeneous platform, the system runs on a number of computers with similar architectures and same operating systems (OSs). In a heterogeneous platform, the architecture and the OS of the nodes are different.

2.2.6. Evaluation method

The performance of cluster system can be evaluated through several metrics to determine the effectiveness of different cluster RMSs. The performance metrics are divided into two main categories, namely system-centric and user-centric evaluation criteria [97]. System centric evaluation criteria depict the overall operational performance of the cluster. Alternatively, user centric evaluation criteria portray the utility achieved by the participants. To assess the effectiveness of RMS system-centric and user-centric criteria, evaluation factors are required. System-centric factors guarantee that system performance is not compromised and user-centric factors assure that desired utility of various RMS are achieved from participant perspective [97]. The system-centric factors can include disk space, access interval, and computing power. User-centric factors can include the cost and execution time of the system. Moreover, a combination of system-centric and user-centric approaches can be used to form another metric that uses features from both, to evaluate the system more effectively [97,105].

2.2.7. Process migration

In cluster, transfer of job from one computer to another without restarting is known as process migration. A standard cluster RMS usually provides the process migration in homogeneous systems. The migration in heterogeneous systems is much more complex because of the numerous complex conversion processes from sources to destination access points.

2.2.8. Correlation of cluster features and resource allocation

All the cluster computing features defined in the previous paragraphs are crucial for an efficient resource management and allocation in the system. The job scheduling policy strictly depends on the type of the job processed in a cluster. Moreover, the job scheduling is completely different for batch (group) scheduling as compared to sequential and simultaneous processed applications. The job processing schema, together with a job structure, mainly determines the speed of the cluster system. The monolithic single-task job processed in a sequential mode is the main reason for possible ineffective system utilization, because some cluster nodes are kept idle for a long time.

The cluster RMSs are defined as a system middleware that provides a single interface for user-level applications to be executed on the cluster. The aforementioned, allows the complexities of the underlying distributed nature of the clusters to be hidden from the users. For effective management, the RMS in cluster computing requires some knowledge of how users value the cluster resources. Moreover, RMS provides support for the users to define QoS requirements for the job execution. In the said scenario, the system-centric approaches have limited abilities to achieve the user desired utility. Therefore, the focus is to increase the system throughput and maximize the resources utilization. The QoS attributes are thoroughly discussed in Section 2.2.

The administration of a centralized RMS is easier than the decentralized structures because a single entity in the cluster has complete knowledge of the system. Moreover, definition of communication protocols for different local job dispatchers is not required. Furthermore, the reliability of centralized systems may be low because of the complete outage of the system in case of central cluster node failure. The distributed administration can tolerate a loss if any node is detached from a cluster. Another important factor of resource allocation in cluster systems is the platform support. In homogenous systems, the resource types are related to the specified scheduling constraints and service requirements defined by the users. The analysis of the requests sent to the system can help in managing the resource allocation process. In heterogeneous platform, the range of resources required, which causes an increase in the complexity of the resource management may vary. A phenomenon where a process, task, or request is permanently denied for resources is known as starvation. To facilitate the parallel processing of applications that requires the same type of resources, most of the cluster systems have homogeneous resource configuration. However, some systems may have heterogeneous resource configurations to achieve concurrent execution of distinct applications that requires different resources [3]. Since the group of nodes in heterogeneous platforms has different resources configurations for specific tasks, we can conclude that the probability of occurring starvation is less in heterogeneous platform as compared to the homogenous platform.

If any node is disconnected from the cluster, then the workload of the node can be migrated to other nodes present in the same cluster. Migration adds the reliability and balancing of resource allocation across the cluster. A single node can request the migration of resources when a request received is difficult to handle. The cluster as a whole is responsible for process migration.

2.3. Grid computer systems: features and requirements

Grid systems are composed of resources that are distributed across various organizations and administrative domains [112]. A grid environment needs to dynamically address the issues involved in sharing a wide range of resources. Moreover, various types of grid systems, such as Computational Grids (CGs), Desktop, Enterprise, and Data Grids, can be designed [99,112]. For each type of grid, a set of various features can be defined and analyzed. We present multiple grid properties that can be extracted from different grid classes to form the generic model of a grid system. A generic model could have all or some of the following extracted properties.

2.3.1. System type

The large ultimate scale of a grid system requires an appropriate architectural model that allows efficient management of geographically distributed resources over multiple administrative domains [7]. The system type can be categorized as computational, data, and service grid, based on the focus of a grid. The computational grid can be categorized as high throughput and distributed computing. The service grid can be categorized as on-demand, collaborative, and multimedia. In hierarchical models the scheduling is a mixture of centralized and decentralized scheduling, having centralized scheduling at the top level and decentralized scheduling at the lower level. Therefore, we categorized our system type into three categories: (a) data, (b) computational, and (c) service.

2.3.2. Scheduling organization

Scheduling organization refers to the way or mechanism that defines the way resources are being allocated. We have considered three main organizations of scheduling namely: (a) centralized, (b) decentralized, and (c) hierarchical. In the centralized model, a central authority has full knowledge of the system. The disadvantages of centralized model are limited scalability, lack of fault tolerance, and difficulty in accommodating local multiple policies imposed by the resource owners.

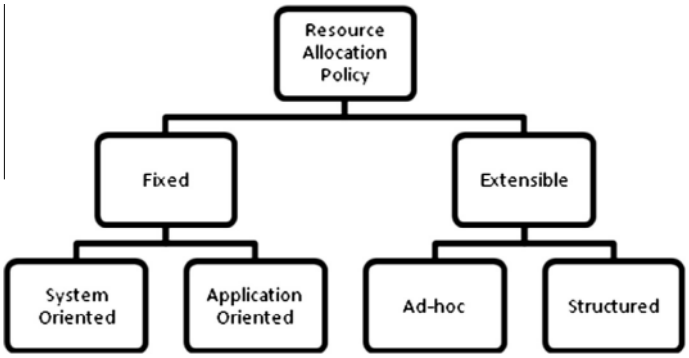


Fig. 7. Taxonomy of resource allocation policies.

Unlike centralized scheduling, the tasks are scheduled by multiple schedulers in decentralized scheduling. Moreover, a single scheduler only maintains the information related to a sub-workflow. Therefore, the model naturally addresses issues, such as fault-tolerance, scalability, site-autonomy, and multi-policy scheduling. The decentralized model is used for large scale network sizes but the scheduling controllers need to coordinate with each other every time for smooth scheduling [136]. The coordination can be achieved through resource discovery or resource trading protocols. Finally, in the hierarchical model, a central meta-scheduler (or meta-broker) interacts with local job dispatchers to define the optimal schedules. The higher-level scheduler manages large sets of resources while the lower level job managers control a small set of resources. The local schedulers have knowledge about resource clusters but cannot monitor the whole system. The advantage of using hierarchical scheduling is the ability to incorporate scalability and fault-tolerance. Moreover, hierarchical scheduling retains some of the advantages of the centralized scheme such as co-allocation (readers are encourage to see [134,135] for more details).

2.3.3. Resource description

Grid resources are spread across the wide-area global network with different local resource allocation policies. The characteristics should include specific parameters needed to express the resource, heterogeneity, structure, and availability in the system. In the NWS project [9], the specification of availability of CPU, TCP Connection establishment time, end-to-end latency, and available bandwidth are needed for resource description. Similarly, Cactus Worm needs an independent service that is responsible for resource discovery and selection based on application-supplied criteria, using GRid Registration Protocol (GRRP) and GRid Information Protocol (GRIP) [10].

2.3.4. Resource allocation policies

A scheduling policy has to be defined for ordering of jobs and requests when any rescheduling is required. Different resource utilization policies are available for different systems due to different administrative domains. Fig. 7 represents the taxonomy of resource allocation policies. Resource allocation polices are necessary for ordering the jobs and requests in all types of grid models. In fixed resource allocation approach, the resource manager implements predefined policy.

Moreover, the fixed resource allocation approach is further classified into two categories namely, system oriented and application oriented. System-oriented allocation policy focuses on maximizing the throughput of the system [63]. The aim of application oriented allocation strategy is to optimize the specific scheduling attributes, such as time and cost (storage capacity). Many examples of systems are available that use application oriented resource allocation policies, such as PUNCH [54], WREN [65], and CONDOR [50]. The resource allocation strategy that allows external agents or entities to change the scheduling policy is called an Extensible Scheduling Policy. The aforementioned can be implemented by using ad hoc extensible schemes that defines an interface used by an agent for the modification of the scheduling policy [63].

2.3.5. Breadth of scope

The breadth of scope expresses the scalability and self-adaptation levels of the grid systems. If the system-or grid-enable application is designed only for specific platform or application, then the breadth of scope is low. Systems that are highly scalable and self-adaptive can be characterized as medium or high breadth of scope. Adopting the self-adaptive mechanisms oriented towards specific type of applications can lead to poor performance of applications not covered by the mechanisms. One example of a breadth of scope is a scheduler that reads applications as if there is no data dependency between the tasks, but if an application has a task dependency, then the scheduler may perform poorly [106].

2.3.6. Triggering information

Triggering information refers to an aggregator service that collects information and check if the data against a set of conditions defined in a configuration file are met [124]. If the conditions are met, then the specified action takes place. The

service plays an important role in notifying certain actions to the administrator or controller whenever any service fails. One example of the aforementioned action is to construct an email notification to the system administrator when the disk space on a server reaches a certain threshold. Triggering information can be used by the schedulers while allocating resources.

2.3.7. System functionality

System functionality is an attribute used to define the core aspect of the system, such as Javelin is a system for Internet-wide parallel computing based on Java.

2.3.8. Correlation of grid features and resource allocation

Grid systems are categorized into various types based on the characteristics of the resource, such as resource type and the allocation policies [67]. The primary focus of computational grids is on processing capabilities. Computational grids are suitable for the execution of compute-intensive and high throughput applications that usually need more computing power by a single resource [67]. Scheduling organization determines the priorities in the resource allocation process. In centralized systems, only single or multiple resources located in a single or multiple domains can be managed [99]. In decentralized scheduling model, the schedulers interact with each other to select resources appropriate for jobs execution. In case of conflicts among resource providers on a global policy for resource management, the aforementioned system (centralized or decentralized) can be difficult to implement as a grid system. The hierarchical system allows remote resource providers to enforce local allocation policies [99]. Several policies are available for resource allocation. The fixed resource allocation policy is generally used for sequentially processed jobs. Extensible policies are used if the application priorities can be set using the external agents.

2.4. Cloud computing systems: features and requirements

The cloud computing systems are difficult to model with resource contention (competing access to shared resources). Many factors, such as the number of machines, types of application, and overall workload characteristics, can vary widely and affect the performance of the system. A comprehensive study of the existing cloud technologies is performed in the following section based on a set of generic features in the cloud systems.

2.4.1. System focus

Each cloud system is designed to focus on certain aspects, such as Amazon Elastic Compute Cloud (EC2) is designed to provide the best infrastructure for cloud computing systems with every possible feature available to the user [55]. Similarly, GENI system [75], focuses on providing a virtual laboratory for exploring future internets in a cloud. Globus Nimbus [114], focuses on extending and experimenting for the set of capabilities, such as resource as an infrastructure and ease of use. Open Nebula [115], provides complete organization of data centers for on-premise Infrastructure as a Service (IaaS) cloud infrastructure.

2.4.2. Services

Cloud computing is usually considered a next step from the grid-utility model [112]. However, the cloud system not only realizes the service but also utilizes resource sharing. Cloud system guarantees the delivery of consistent services through advanced data centers that are built on compute and storage virtualization technologies [72,112]. The type of services that a system provides to a user is an important parameter to evaluate the system [11]. Cloud computing is all about providing services to the users, either in the form of SaaS, PaaS, or IaaS. The cloud computing architecture can be categorized based on the type of services they provide, such as Amazon EC2 provides computational services and storage services. However, the Sun Network.com (Sun Grid) only provides computational services. Similarly, we can categorize Microsoft Live Mesh and GRIDS Lab Aneka as infrastructure and software based cloud, respectively.

2.4.3. Virtualization

Cloud resources are modeled as virtual computational nodes connected through large-scale network conferring to the specified topology. Peer-to-peer ring topology is a commonly used example for a cloud resource system and users community organization [72]. Based on the virtualization, the cloud computing paradigm allows workloads to be deployed and scaled-out quickly through the rapid provisioning of Virtual Machines (VMs) on physical machines. We evaluated a number of systems based on the entities or the processes responsible for performing virtualization.

2.4.4. Dynamic QoS negotiation

Real-time middleware services must guarantee predictable performance under specified load and failure conditions [105]. Provision of QoS attributes dynamically at run-time based on specific conditions is termed as Dynamic QoS negotiation, such as renegotiable variable bit-rate [129]. Moreover, dynamic QoS negotiations ensure graceful degradation when the aforementioned conditions are violated. QoS requirements may vary during the execution of the system workflow to allow the best adaptation to customer expectations. Dynamic QoS negotiation in cloud systems are performed by either a dedicated process or by an entity. Moreover, self-algorithms [105] can also be implemented to perform dynamic QoS negotiation. In Eucalyptus group managers [71], the dynamic QoS operations are performed by the resource services. Dynamic QoS

negotiation provides more flexibility to the cloud that can make a difference while selecting two different clouds, based on the requirements of the user. That is the reason Dynamic QoS is used as a comparison attribute in the cloud.

2.4.5. User access interface

User access interface defines the communication protocol of the cloud system with general user. Access interfaces must be equipped with the relevant tools necessary for better performance of the system. Moreover, the access interface can be designed as a command line, query based, console based, or graphical form interface. Although the access interface is available in cluster and grid but in case of cloud, the access interface is important because if the interface provided to the user is not user-friendly, then the user might not use the service. In another scenario, suppose two cloud service providers provides same services but if one has a user-friendly interface and the second one does not, then user would definitely prefer the one with a user-friendly interface. In such scenarios the access interface plays an important role and that is why it is used as a comparison feature under cloud.

2.4.6. Web APIs

In cloud, the Website Application Programming Interface (Web API) is a web service dedicated for the combination of multiple web services into new applications [11]. A set of Hypertext Transfer Protocol (HTTP) request messages and description of the schema of response messages in JavaScript Object Notation (JSON) or Extensible Markup Language (XML) format makes a web API [11]. The ingredients of current Web APIs are Representational State Transfer (REST) style communication. Earlier APIs were developed using Simple Object Access Protocol (SOAP) based services [11].

2.4.7. Value added services

Value added services are defined as additional services beyond the standard services provided by the system. Value added services are available for a modest additional fee (or for free) as an attractive and low-cost alternative system support. Moreover, the purpose of value added services are to: (a) promote the cloud system, (b) attract the new service users, and (c) keep the old service users intact. The services mentioned in SLA are standard services. Moreover, the services that are provided to end-users to promote the standard services come under the category of value added services. Value added services are important to promote the cloud and to provide an edge over competitors. If one cloud is only offering SLA based services and other is offering SLA based service plus value added services too, then generally end-users will prefer the cloud that provides both of the services.

2.4.8. Implementation structure

Different programming languages and environments are used to implement a cloud system. The implementation package can be monolithic and consists of a single specific programming language. The Google app engine is an example of a cloud system that has been implemented in the Python Script Language. Another class is the high-level universal cloud systems, such as Sun Network.com (Sun Grid) that can be implemented using Solaris OS and programming languages like Java, C, C++, and FORTRAN. Implementation structure is an important aspect to compare amongst different clouds because if a cloud is implemented in a language that is obsolete, then people will hesitate using such cloud.

2.4.9. VM migration

The VM technology has emerged as a building block of data centers, as it provides isolation, consolidation, and migration of workload. The purpose of migrating VM is to seek improvement in performance, fault tolerance, and management of the systems over the cloud. Moreover, in large scale systems the VM migration can also be used to balance the systems by migrating the workload from overloaded or overheated systems to underutilized systems. Some hypervisors, such as Vmware [89] and Xen, provides "live" migration, where the OS continues to run while the migration is performed. VM migration is an important aspect of the cloud towards achieving high performance and fault tolerance.

2.4.10. Pricing model in cloud

The pricing model implemented in the cloud is pay-as-you-go model, where the services are charged as per the QoS requirements of the users. The resources in the cloud, such as network bandwidth and storage, are charged on a specific rate. For example, the standard price for block storage on HP cloud is $0.10 per GB/mo [138]. The prices of the clouds may vary depending on the types of services they provide.

2.4.11. Correlation of cloud features and resource allocation

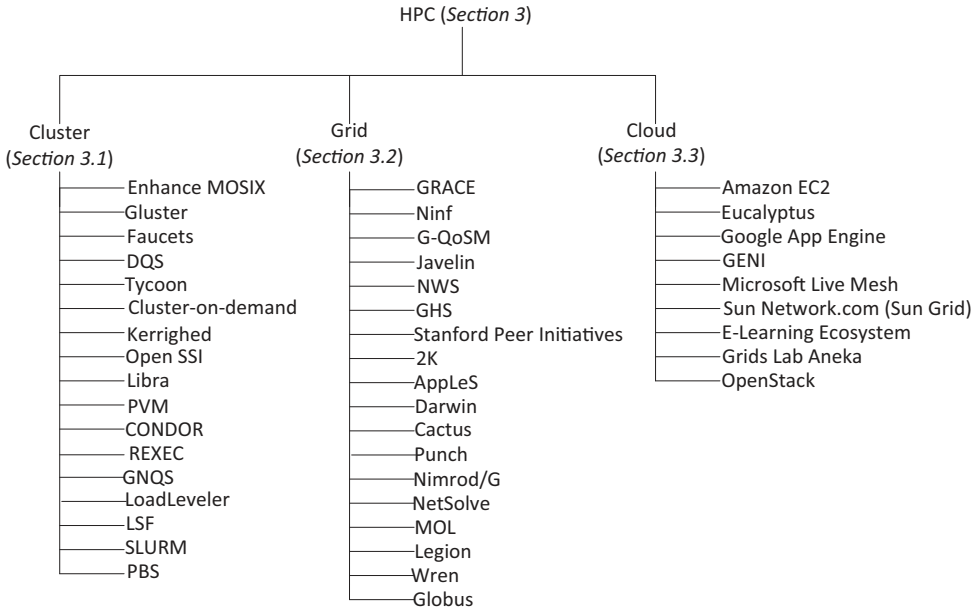
The focus of the cloud system is an important factor for the selection of appropriate resources and services for the cloud users. Some resources may require a specific type of infrastructure or platform. However, the cloud computing is more service-oriented than resource-oriented [17]. The cloud users do not care much about the resources, but are more concerned with the services being provided. Virtualization is used to hide the complexity of the underlying system and resources. User satisfaction is one of the main concerns in provisioning cloud computing web services Dynamic QoS negotiations can only be made if the resources are available.

3. Mapping the hpc systems classification to various cluster, grid and cloud systems: comparison and survey of the existing HPC solutions

In Table 1, a comparison of three HPC categories (cluster, grid, and cloud) is provided. We classify various HPC research projects and commercial products according to the HPC systems classification that we have developed in Section 2. The list of the systems discussed is not exhaustive but is representative of the classes. The projects in each category have been chosen based on the factors specified for each HPC class that was reported in Sections 2.2–2.4.

3.1. Cluster computing system

A list of representative cluster projects and a brief summary is provided in Table 2. The systems discussed below are characterized based on the generic cluster system features highlighted in Section 2.2.



3.1.1. Enhanced MOSIX

Enhanced Mosix (E-Mosix) is a tailored version of Mosix project [13], which was geared to achieve efficient resource utilization amongst nodes on a distributed environment. Multiple processes are created by the users to run the applications. Mosix will then discover the resources and will automatically migrate the processes among the nodes for performance improvement without changing the run-time environment of the processes. E-Mosix uses cost-based policy for process migration. The node in every cluster makes the resource allocation decisions independently. Different resources are collected and the overall system performance measure is defined as a total cost of the resource utilization. E-Mosix supports parallel job processing mode. Moreover, migration process is used to decrease the overall cost of job execution on different machines in the cluster. Furthermore, a decentralized resource control is implemented and each cluster node in the system is supplied with an autonomous resource assignment policy.

3.1.2. Gluster

Gluster defines a uniform computing and storage platform for developing applications inclined towards specific tasks, such as storage, database clustering, and enterprise provisioning [14]. The distribution of the Gluster is independent and has been tested on a number of distributions. Gluster is an open source and scalable platform whose distributed file system (GlusterFS) is capable of scaling up to thousands of clients. Commodity servers are combined with Gluster and storage to form a massive storage networks. Gluster System Provisioning (GlusterSP) and GlusterHPC are bundled cluster applications associated with Gluster. The said system can be extended using Python scripts [14].

3.1.3. Faucets

Faucets [116] is designed for processing parallel applications and offers an internal adaptation framework for the parallel applications based on adaptive Message Passing Interface (MPI) [16] and Charm++ [109] solutions. The number of

Table 2
Comparison of cluster computing systems.

System	Job processing type	QoS attributes	Job composition	Resource allocation control	Platform support	Evaluation method	Process migration
Enhanced MOSIX [12]	Parallel	Cost	Single task	Decentralized	Heterogeneous	User-centric	Yes
Gluster [14]	Parallel	Reliability (no point of failure)	Parallel task	Decentralized	Heterogeneous	N/A	Yes
Faucets [116]	Parallel	Time, cost	Parallel task	Centralized	Heterogeneous	System-centric	Yes
DQS [19]	Batch	CPU memory sizes, hardware architecture and OS versions.	Parallel Task	Decentralized	Heterogeneous	System-centric	No
Tycoon [20]	Sequential	Time, cost	Multiple task	Decentralized	Heterogeneous	User-centric	No
Cluster-on-demand [22]	Sequential	Cost in terms of time	Independent	Decentralized	Heterogeneous	User-centric	No
Kerrighed [23,24]	Sequential	Ease of use, high performance, high availability, efficient resources management, and high customizability of the OS	Multiple task	Decentralized	Homogeneous	System-centric	Yes
OpenSSI [25]	Parallel	Availability, scalability and manageability	Multiple task	Centralized	Heterogeneous	System-centric	Yes
Libra [26]	Batch, sequential	Time, cost	Parallel	Centralized	Heterogeneous	System-centric, User-centric	Yes
PVM [28]	Parallel, concurrent	Cost	Multiple task	Centralized	Heterogeneous	User-centric	Yes
Condor [50,51]	Parallel	Throughput, productivity of computing environment	Multiple task	Centralized	Platform support	System-centric	Yes
REXEC [30]	Parallel, sequential	Cost	Independent, single task	Decentralized	Homogeneous	User-centric	No
GNQS [31]	Batch, parallel	Computing power	Parallel processing	Centralized	Heterogeneous	System-centric	No
LoadLeveler [32]	Parallel	Time, high availability	Multiple task	Centralized	Heterogeneous	System-centric	Yes
LSF [90]	Parallel, Batch	Job submission simplification, setup time reduction and operation errors	Multiple task	Centralized	Heterogeneous	System-centric	Yes
SLURM [91]	Parallel	Simplicity, scalability, portability and fault tolerance	Multiple task	Centralized	Homogeneous	System-centric, User-centric	No
PBS [92]	Batch	Time, jobs queuing	Multiple task	Centralized	Heterogeneous	System-centric	Yes

applications executed on Faucets can vary [18]. The aforementioned process allows the utilization of all resources that are currently available in the system. For each parallel task submitted to the system, the user has to specify the required software environment, expected completion time, number of processors needed for the task completion, and budget limits. The privileged scheduling criterion in Faucets is the completion time of a job. The total cost of the resource utilization calculated for a particular user is specified based on the bids received from the resource providers. Faucets supports time-shared scheduling that simultaneously executes adaptive jobs based on dissimilar percentages of allocated processors. Faucets support parallel job processing type. Moreover, the constraints about the requirements of any parallel task remain constant throughout the task execution. Jobs are submitted to Faucets with a QoS requirement and subscribing clusters return bids. Moreover, the best bid that meets all criteria is selected. Bartering is an important unit of Faucet that permits cluster maintainers to exchange computational power with each other. Moreover, units are awarded when the bidding cluster successfully runs an application. Users can later on trade the bartering units to use the resources on other subscribing clusters.

3.1.4. Distributed Queuing System (DQS)

DQS is used for scheduling background tasks to a number of workstations. The tasks are presented to the system as a queue of applications. The queue of tasks is automatically organized by the DQS system based on the current resource status [19]. Jobs in the queue are sorted on the priority of subsequent submission pattern, internal sub-priority, and the job identifier. The sub-priority is calculated each time the master node scans the queued jobs for scheduling. The calculations are relative to each user and reflect the total number of jobs in the queue that are ahead of each job. The total number of jobs includes any of the use jobs that are in "Running" state or in "Queued" state.

3.1.5. Tycoon

Tycoon allocates cluster resources with different system performance factors, such as CPU cycles, memory, and bandwidth [20,21]. Tycoon is based on the principle of proportional resource sharing. Moreover, the major advantage of Tycoon is to differentiate the values of the jobs. Communication delay is a major factor for resource acquisition latency, and no process of manual bidding is available in Tycoon. Manual bidding supports proficient use of different resources when no precise bids are present at all. Tycoon is composed of four main components namely: (a) bank, (b) auctioneers, (c) location service, and (d) agents. Tycoon uses two-tier architecture for the allocation of resources. Moreover, Tycoon differentiates between allocation mechanism and user strategy. Allocation mechanism offers different means to seek user assessments for efficient execution and user strategy captures high-level preferences that vary across number of users but are more application-dependent. The division of allocation mechanism and user strategy permits requirements not to be restricted and dependent.

3.1.6. Cluster on demand

Cluster-on-demand allocates servers from a common pool to multiple partitions called virtual clusters, with independently configured software environments [22]. The jobs executed in the system are implicitly single-task applications and are ordered on the basis of arrival time. For each job submitted to the system, the user specifies a value function containing a constant reduction factor for the required level of services needed by the user [22]. The value function remains static throughout the execution once the agreement has been approved by the user. A cluster manager is responsible for scheduling the tasks to resources from different administrative domains. The support of adaptive allocation of resource update is available. Moreover, the support in cluster manager forces less costly dedicated jobs to wait for more costly new tasks that may arrive later in the future. A deduction can be made from the aforementioned, that no hard constraints are supported because many accepted jobs can take more time for the completion than anticipated. The cost measure of cluster-on-demand is the cost of node configuration for a full wipe clean install. The cluster-on-demand uses a user-centric evaluation of cost measure and the major cost factor is the type of hardware devices used.

3.1.7. Kerrighed

Kerrighed is a cluster system with a Linux kernel patch as a main module for controlling the whole system behavior [23,24]. For fair load balancing of the cluster, schedulers use sockets, pipe, and char devices. Moreover, the use of devices does not affect the cluster communication mechanisms due to seamless migration of the applications across the system. Furthermore, the migration of single threaded and multi-threaded applications are supported in the process. The running process at one node can be paused and restarted at another node. Kerrighed system provides a view of single Symmetric Multi-Processing (SMP) machine.

3.1.8. Open Single System Image (OpenSSI)

OpenSSI is an open source uniform image clustering system. Moreover, the collection of computers to serve as a joint large cluster [25] is also supported by OpenSSI. Contrary to Kerrighed, in OpenSSI, the number of resources that are available may vary during the task execution. OpenSSI is based on Linux OS. The concept of bit variation process migration that is derived from Mosix, is used in OpenSSI. Bit Variation dynamically balances the CPU load on the cluster by migrating different threaded processes. The process management in OpenSSI is tough. A single process ID is assigned to each process on a cluster and the inter process communication is handled cluster wide. The limitation of OpenSSI is the support of maximum 125 nodes per cluster.

3.1.9. Libra

Libra takes advantage of the number of jobs based on the system and user requirements [26]. Different resources are allocated based on the budget and deadline constraints for each job. Libra communicates with the federal resource manager that is responsible for collecting information of different resources presented in the cluster. In case of a mixed composition of resources, estimated execution time is calculated on diverse worker nodes. Libra assigns different resources to executing jobs based on the deadlines. A centralized accounting mechanism is used for resource utilization of current jobs, to periodically relocate time partitions for each critical job and to meet the deadlines. Libra assumes that each submitted job is sequential and is composed of a single task. Libra schedules tasks to internal working nodes available in the cluster [27]. Each internal node has a task control component that relocates and reassigns processor time, and performs partitioning periodically based on the actual execution and deadline of each active job. The system evaluation factors to assess overall system performance are system-centric with average waiting and response time as the parameters. However, Libra performs better than traditional First Come First Serve (FCFS) scheduling approach for both user-centric and system-centric evaluation factors.

3.1.10. Parallel Virtual Machine (PVM)

PVM is a portable software package combining a heterogeneous collection of computers in a network to provide a view of a single large parallel computer. The aim of using PVM is to aggregate memory and power of many computers to solve large computational problems in a cost efficient way. To solve much larger problems, PVM accommodates existing computer hardware with some minimal extra cost. A PVM user outside the cluster can view the cluster as a single terminal. All cluster details are hidden from the end user, irrespective of how cluster puts tasks on individual nodes. PVM is currently being used by

a number of websites across the globe for solving medical, scientific, and industrial problems [28]. PVM is also employed as an educational tool for teaching parallel programming courses.

3.1.11. Rexec

In REXEC, a resource sharing mechanism, the users struggle for shared resources in a cluster [30]. The computational load of the resources is balanced according to the total allocation cost, such as credits per minute that users agreed to pay for resource utilization. Multiple daemons select the best node to execute particular tasks that are the key components of the decentralized resource management control system. Numbers of jobs are mapped to the distributed resources at the same time intervals according to the time-shared scheduling rules. REXEC supports parallel and sequential job processing. Users specify the cost restrictions that remains fixed after the task submission. The resource assignments already presented in the system are reassigned, whenever a new task execution is being initialized or finished. REXEC uses an aggregate utility function as a user-centric evaluation factor that represents the cost of all tasks on the cluster. The end-users are charged based on the completion times of the tasks.

3.1.12. Generic Network Queuing System (GNQS)

GNQS is an open source batch processing system. The networks of computers or applications on a single machine are scheduled through GNQS that does not allow tasks to be executed simultaneously [31]. GNQS is not a shareware application, works in GNU OS, and is maintained by a large community across the Internet. ANSI-C language is required to compile the code with root privileges to successfully run the GNQS on a single local computer.

3.1.13. Load Leveler

Load Leveler [32] is a parallel scheduling system developed by IBM that works by matching the processing needs of each task and priorities of available resources. Number of end users can execute jobs in a limited time interval by using load leveler. For high availability and workload management, the Load Leveler provides a single point of control. In a multi-user production environment, the use of Load Leveler supports aggregate improvement in system performance as well as turn-around time with equal distribution of the resources. In Load Leveler, every machine that contributes must run one or more daemons [32].

3.1.14. Load Sharing Facility (LSF)

LSF [90] has a complete set of workload management abilities that manages workload in distributed, demanding, and critical HPC environments. LSF executes batch jobs. The set of workload management and intelligent scheduling features fully utilize the computing resources. LSF schedules a complex workload and provides a highly available and scalable architecture. Ref. [90], provides HPC components like HPC data center for managing workload and also provides vendor support.

3.1.15. Simple Linux Utility for Resource Management (SLURM)

SLURM [91] is an open source, scalable, and fault tolerant cluster management and job scheduling system. SLURM is used in small and large Linux cluster environments. SLURM provides exclusive and non-exclusive access of computing resources to users. Then, the execution and monitoring of the allocated computing resources are performed. Finally, the awaiting requests are accomplished.

3.1.16. Portable Batch System (PBS)

PBS [92] provides job resource management in batch cluster environments. In HPC environments, PBS provides the jobs information to the Moab, which is a job scheduler used in PBS. Moab decides the selection of jobs for execution. PBS selects and dispatches jobs from the queue to cluster nodes for execution. PBS supports non-interactive batch jobs and interactive batch jobs. The non-interactive jobs are more common. The essential execution commands and resource requests are created in the form of a job script that is submitted for execution.

3.1.17. Condor (HTCondor)

A large collection of heterogeneous machines and networks are managed by Condor (recently renamed as HTCondor) high throughput computing environment [50,51]. Condor shares and combines the idle computing resources. Condor reserves the information of the originating machine specifications through remote system call capabilities. The remote system call tracks the originated machines when the file system or scheme is not shared among the users. A Condor matchmaker is used to determine the compatible resource request. The matchmaker triggers a query to the condor collector for resource information stored for resource discovery.

3.2. Grid computing system

A diverse range of applications is employed in computational grids. Scientists and engineers rely on grid computing to solve challenging problems in engineering, manufacturing, finance, risk analysis, data processing, and science [33]. Table 3 shows the representative grid systems that are analyzed according to the grid features specified in Section 2.3. All the values

Table 3
Comparison of grid computing systems.

System	Sys type	Scheduling organization	Resource description	Resource allocation policy	Breadth of scope	Triggering info	Sys functionality
GRACE [33,34]	Computational	Not specified can be decentralized/ad-hoc	CPU process power, memory, storage capacity and network bandwidth	Fixed AOP	High	High	Resources are allocated on demand and supply
Ninf [35]	Computational	Decentralized	No QoS, periodic push dissemination, centralized queries discovery	Fixed AOP	Medium	Low	A global computing client–server based system
G-QoSM [37,38,119]	On-demand	Requirements matching	Processing power and bandwidth	Fixed AOP	High	High	SLA based resources are allocated in the system
Javelin [40]	Computational	Decentralized	Soft QoS, distributed queries discovery, other network directory store, periodic push dissemination	Fixed AOP	Low	Medium	A system for Internet-wide parallel computing based on Java
NWS [47]	Hierarchical	Heuristics based host capacity	End-to-end latency and available bandwidth, availability of CPU	N/A	Low	Low	Used for short term performance prediction
GHS[41]	Hierarchical	Heuristics based on host capacity	Availability of CPU, TCP connection establishment time, end-to-end latency	N/A	Medium	Medium	Scalability and precision in prediction at high level than NWS [47]
Stanford peer initiatives	Computational	Hierarchical/ decentralized	CPU cycles, disk space, network bandwidth	NA	High	Medium	Distribution of main costs of sharing data, disk space for storing files and bandwidth for transfer
2 K [43,44]	On-demand	Hierarchical/ decentralized	Online dissemination, Soft network QoS, agent discovery	Fixed AOP	High	Medium	Flexible and adaptable distributed OS used for a wide variety of platforms
AppLeS [48,36]	High-throughput	Hierarchical/ decentralized	Models for resources provided by Globus, Legion, or Netsolve	Fixed AOP	Low	Medium	Produces scheduling agents for computational grids
Darwin [49]	Multimedia	Hierarchical/ decentralized	Hard QoS, graph namespace	Fixed system oriented policy (SOP)	Low	High	Manages resources for network services
Cactus Worm [52]	On-demand	Requirements matching	N/A	Fixed AOP	High	Medium	When required performance is not achieved the system allows applications to adapt accordingly
PUNCH [54]	Computational	Hierarchical/ decentralized	Soft QoS, periodic push dissemination, distributed queries discovery	Fixed AOP	Medium	Medium	A middleware that provides transparent access to remote programs and resources.
Nimrod/G [56,57]	High-throughput	Hierarchical/ decentralized	Relational network directory data store, soft QoS, distributed queries discovery	Fixed AOP	Medium	Medium	Provides brokering services for task farming application
NetSolve [60]	Computational	Decentralized	Soft QoS, periodic push dissemination, distributed queries discovery	Fixed AOP	Medium	Medium	A network-enabled application server for solving computational problems in distributed environment
MOL [61]	Computational	Decentralized	Distributed queries discovery, periodic push dissemination	Extensible ad hoc scheduling policies (ASP)	Low	Low	Provide resource management for dynamic communication, fault management, and access provision
Legion [63,64](1999)	Computational	Hierarchical/ decentralized	Soft QoS, periodic pull dissemination, distributed queries discovery	Extensible structured scheduling policy (SSP)	Medium	Medium	Provides an infrastructure for grid based on object meta system
Wren [65]	Grid	No mechanism for initial scheduling	NA	Fixed AOP	Low	Low	Provide active probing with low overhead.

Table 3 (*continued*)

System	Sys type	Scheduling organization	Resource description	Resource allocation policy	Breadth of scope	Triggering info	Sys functionality
Globus [66]	Hierarchical	Decentralized	Soft QoS, network directory store, distributed queries discovery	Extensible Ad-hoc scheduling policy (ASP)	High	Medium	Provides basic services for modular deployment of grids in Globus meta computing Toolkit

of the features are straight forward. For some features, we did a comparative study, such as breadth of scope or triggering information and the values are high, medium, or low. No threshold value for the categorization is provided.

3.2.1. Grid Architecture for Computational Economy (GRACE)

GRACE is a generic infrastructure for the market-based grid approach that co-exists with other grid systems, such as Globus. The interactions of the grid users with the system are provided through grid resource broker (GRB). GRACE employs Nimrod-G grid scheduler [33] responsible for: (a) resource discovery, (b) selection, (c) scheduling, and (d) allocation. The resource brokers fulfill the user demands by optimizing execution time of the jobs and user budget expenses, simultaneously [33]. GRACE architecture allocates resources on supply and demand basis [34]. The resources monitored by GRACE are software applications or hardware devices. GRACE enables the control of CPU power, memory, storage capacity, and network bandwidth. The resources are allocated according to the fixed application-oriented policy.

3.2.2. Network infrastructure (Ninf)

Ninf [35] is an example of a computational grid that is based on a client–server infrastructure. Ninf clients are connected with the servers through local area networks. The server machine and the client machines could be heterogeneous, and the data to be communicated is translated into a mutual network data format [35]. The components of the Ninf system are client interfaces, remote libraries, and a meta-server. Ninf applications invoke Ninf libraries and the request is forwarded to the Ninf meta-server that maintains the Ninf servers directory. Ninf meta-server forwards the library request to the appropriate server. Moreover, Ninf uses centralized resource discovery mechanism. The computational resources are registered with meta-server through library services [36]. The scheduling mechanism in Ninf is decentralized and the server performs actual scheduling of the client requests. Ninf uses a fixed application oriented policy, has a medium level breadth of scope, and provides no QoS. The triggering information in Ninf is low.

3.2.3. Grid-Quality of Services Management (G-QoSM)

G-QoSM [34,37,38] system works under an Open Grid Service Architecture (OGSA) [119]. G-QoSM provides resource and service discovery support, based on QoS features. Moreover, guarantee of supporting QoS at application, network, and middle grid level is also provided. G-QoSM provides three levels of QoS namely: (a) best effort, (b) controlled, and (c) guaranteed levels. The resources are allocated on the basis of SLA between the users and providers. G-QoSM utilizes SLA mechanism, so the triggering information as well as breadth of scope is high. The scheduling organization in G-QoSM can be centralized or decentralized. However, the main focus of G-QoSM is managing the QoS. The resources focused by the G-QoS management are the bandwidth and processing power. G-QoSM uses fixed application oriented policy for resource allocation.

3.2.4. Javelin

Javelin is a Java based infrastructure [40] that may be used as an Internet-wide parallel computing system. Javelin is composed of three main components: (a) clients, (b) hosts, and (c) brokers. Hosts provide computational resources, clients seek for computational resources and resource brokers support the allocation of the resources. In Javelin, hosts can be attached to a broker, considering as a resource. Javelin uses hierarchical resource management [36]. If a client or host wants to connect to Javelin, then a connection with Javelin broker has to be made that is agreed to support the client or host. The backbone of Javelin is the Broker Name Service (BNS). The BNS is an information system that keeps the information about available brokers [40]. Javelin has a decentralized scheduling organization with a fixed application oriented resource allocation policy. The breadth of scope of Javelin is low and only supports Java based applications. The triggering information of Javelin is medium.

3.2.5. Network weather service (NWS)

NWS [47] is a distributed prediction system for the network (and resources) dynamics. The prediction mechanism in NWS is based on the adaptation strategies that analyze the previous system states. In NWS, system features and network performance factors, such as bandwidth, CPU speed, TCP connection establishment time, and latency are considered as the main criteria of resource description measurements [34]. System, such as NWS have been used successfully to choose between replicated web pages [127] and to implement dynamic scheduling agents for meta-computing applications [125,126].

Extensible system architecture, distributed fault-tolerant control algorithms, and adaptive programming techniques has been illuminated by the implementation of the NWS to operate in a variety of meta-computing and distributed environments with changing performance characteristics. NWS uses host capacity based scheduling organization. Moreover, NWS works well for short time processes in restricted area grid clusters. Therefore, the breadth of scope and triggering information of NWS is low.

3.2.6. Grid harvest service (GHS)

The goal of GHS is to achieve high scalability and precision in a network [41]. The GHS system is comprised of five subsystems, namley: (a) task allocation module, (b) execution management system, (c) performance measurement, (d) performance evaluation, and (e) task scheduling modules [42]. GHS enhances application performance by task re-scheduling and utilizing two scheduling algorithms. First algorithm minimizes task execution time and the second algorithm is used to assign the tasks to an individual resource. GHS, like NWS [47], uses host capacity based heuristics as scheduling organization. The breadth of scope and the triggering information is of medium level in GHS [34].

3.2.7. Stanford Peers Initiative

Stanford Peers Initiative utilizes a peer-to-peer data trading framework to create a digital archiving system. Stanford Peers Initiative uses a unique bid trading auction method that seeks bids from distant web services to replicate the collection. In response, each remote web service replies that reflect the amount of total disk storage space [29]. The local web service selects the lowest bid for maximizing the benefits. Because the system focuses on preserving the data for the longest possible period, the major system performance factor is the reliability. The reliability measure is a Mean Time To Failure criterion (MTTF) for each local web service. Each web service tries to minimize the total cost of trading that is usually measured in terms of disk space provided. For replicating data collection, a decentralized management control is implemented in the system. The web service makes the decision to select the suitable remote services. Each web service is represented as an independent system entity. The storage space remains fixed throughout, even if a remote service is selected.

3.2.8. 2k

2 K grid system provides distributed services for multiple flexible and adaptable platforms [43,44]. The supported platform ranges from Personal Digital Assistants (PDAs) to large scale computers for the application processing. 2 K is a reflective OS built on top of a reflective Object Request Broker (ORB), dynamicTAO [45], a dynamically configurable version of The Ace ORB (TAO) [46]. The key features of 2 K are: (a) distribution, (b) user-centrism, (c) adaptation, and (d) architectural awareness. 2 K is an example of an on-demand grid system that uses agents for resource discovery and mobile agents to perform resource dissemination functionality [36]. 2 K uses decentralized and hierarchical scheduling organization and a fixed application oriented resource allocation policy. In 2 K system, no mechanism for rescheduling is supported. The breadth of scope of the 2 K system is high due to multiple range of platforms. Moreovoer, the triggering information is medium due to soft QoS provisioning.

3.2.9. AppLeS

AppLeS is an application level grid scheduler that operates as an agent in a dynamic environment. AppLeS assists an application developer by enhancing the scheduling activity. For each application, an individual AppLeS agent is designed for resource selection. Unlike the projects, such as Legion or Globus grid packages [48] that utilize resource management systems, AppLeS agents are not utilized as resource management system. However, AppLeS is used for computational purposes. AppLeS provide templates that are used in structurally similar applications. AppLeS utilize hierarchical or decentralized schedulers and a fixed application oriented policy for resource allocation [36]. The triggering information in AppLeS is medium and the breadth of scope is low.

3.2.10. Darwin

Darwin is a grid resource management system that provides value-added network services electronically [49]. The main features of the system are: (a) high level resource selection, (b) run-time resource management, (c) hierarchical scheduling, and (d) low level resource allocation mechanisms [49]. Darwin utilizes hierarchical schedulers and online rescheduling mechanisms. The resource allocation policy in Darwin is fixed system-oriented. To allocate resources globally in grid, The system employs a request broker called Xena [49]. For resource allocation at higher level, Darwin uses a hierarchical fair service curve scheduling algorithm (H-FSC). Darwin runs in routers and provides hard network QoS.

3.2.11. Cactus Worm

Cactus Worm [52] is an on-demand grid computing system. Cactus Worm supports an adaptive application structure and can be characterized as an experimental framework that can handle dynamic resource features. Cactus supports dynamic resource selection for resource interchange through migration. The migration mechanism is performed only when the performance is not adequate [52]. Cactus Worm supports different architectures namely: (a) uni-processors, (b) clusters, and (c) supercomputers [53]. The scheduling organization in Cactus Worm is based on requirements matching (Condor) and uses a fixed-based application oriented policy (AOP) for resource allocation. The functionality of the Cactus Worm is

Table 4
Comparison of cloud computing systems.

System	System focus	Services	Virtualization	Dynamic QoS negotiation	User access interface	Web APIs	Value added services	Implementation structure
Amazon elastic compute cloud (EC2) (2006)	Infrastructure	Compute, storage (Amazon S3)	OS level running on a Xen hypervisor	None	EC2 command-line tools	Yes	Yes	Customizable Linux-based AMI
Eucalyptus (2009)	Infrastructure	Compute, storage	Instance manager	Group managers through resource services	EC2's SOAP and query interfaces	Yes	Yes	Open source Linux-based
Google App Engine (2008)	Platform	Web application	Application container	None	Web-based administration console	Yes	No	Python
GENI (2007)	Virtual laboratory	Compute	Network accessible APIs	Clearing house based resource allocation	Slice federation architecture 2.0	Network accessible APIs	Yes	SFA (PlanetLab), ProtoGENI and GCF
Microsoft Live Mesh (2005)	Infrastructure	Storage	OS level	None	Web-based live desktop and any devices with live mesh installed	N/A	No	N/A
Sun Network.com (Sun Grid) (2007)	Infrastructure	Compute	Job management system (Sun Grid Engine)	None	Job submission scripts, sun grid web portal	Yes	Yes	Solaris OS, Java, C, C++, FORTRAN
E-learning ecosystem (2007)	Infrastructure	Web application	Infrastructure layer	None	Web-based dynamic interfaces	Yes	Yes	Programming models available in ASP.Net for front end and any database like SQL, Oracle at the Back end
GRIDS Lab Aneka (2008)	Software platform for enterprise clouds	Compute	Resource manager and scheduler	SLA-based resource reservation on Aneka Side	Workbench, web-based portal	Yes	No	APIs supporting different programming models in C# and .Net supported
OpenStack (2011)	Software platform	Compute, storage, web Image	Compute, web image service	None	REST interface	Yes	Yes	N/A

expressed in adaptation of the resource allocation policy if the required performance level is not achieved. Moreover, the breadth of scope is high and triggering information is at the middle level in Cactus Worm.

3.2.12. Punch

Purdue University Network Computing Hub (PUNCH) is a network-based computing middleware testbed that provides OS services in a distributed computing environment [54]. PUNCH is a multi-user and multi-process environment that allows: (a) a transparent remote access to applications and resources, (b) access control, and (c) job control functionality. PUNCH supports a virtual grid organization by fully decentralized and autonomous management of resources [110]. The key concept is to design and implement a platform that provides independence between the applications and the computing infrastructure. PUNCH possesses hierarchical decentralized resource management and predictive machine learning methodologies for mapping the jobs to resources. PUNCH uses: (a) an extensible schema model, (b) a hybrid namespace, (c) soft QoS, (d) distributed queries discovery, and (e) periodic push dissemination as resources. The resources are allocated according to the fixed application oriented policy. The functionality of PUNCH systems is expressed in terms of flexible remote access to the user and the computing infrastructure of the application. The breadth of scope and triggering information of PUNCH are at the middle level.

3.2.13. Nimrod/G

Nimrod/G is designed to seamlessly execute large-scale parameter study simulations, such as parameter sweep applications, through a simple declarative language and GUI on computational Grids [56,57]. Nimrod/G is a grid resource broker for managing and steering task farming applications and follows a computational market-based model for resource management. Nimrod/G strives for low cost access to computational resources using GRACE services. Moreover, the user defined

constraints cost is minimized using adaptive scheduling algorithms. Beside the parameter studies, Nimrod/G also provides support for a single window to: (a) manage and control experiments, (b) discover resources, (c) trade resources, and (d) perform scheduling [58,66]. Nimrod/G uses a task performing engine that generates user defined scheduling policies. Nimrod uses hierarchical decentralized scheduler and predictive pricing models as scheduling organizations. Moreover, Nimrod/G uses resource descriptions, such as (a) relational network directory data store, (b) soft QoS, (c) distributed queries discovery, and (d) periodic dissemination. Fixed application-oriented policy driven by user-defined requirements, such as deadline and budget limitations are used in Nimrod/G for resource allocation. Active Sheets is an example of Nimrod/G used to execute Microsoft Excel computations/cells on the Grid [59].

3.2.14. NetSolve

NetSolve [60] is an application server based on a client–agent–server environment. NetSolve integrates distributed resources to a desktop application. NetSolve resources include hardware, software, and computational software packages. TCP/IP sockets are used for the interaction among the user, agents, and servers. The server can be implemented in any scientific package. Moreover, the clients can be implemented in C, FORTRAN, MATLAB, or web pages. The agents are responsible for locating the best possible resources available in the network. Once the resource is selected, the agents execute the client request and return the answers back to the user. NetSolve is a computational grid with a decentralized scheduler. NetSolve uses soft QoS, distributed queries discovery and periodic push dissemination for the resource description. Moreover, a fixed application oriented policy is used for resource allocation. Breadth of scope and triggering information of NetSolve is medium as scalability is limited to certain applications.

3.2.15. Meta Computing Online (MOL)

MOL system consists of a kernel as the core component of system and provides the basic infrastructure for interconnected resources, users, and third party meta-computer components [61]. MOL supports dynamic communications, fault management, and access provisions. The key aspects of MOL kernel are reliability and flexibility. Moreover, MOL is the first meta-computer infrastructure that does not reveal a single point of failure [62]. MOL has a decentralized scheduler and uses: (a) hierarchical namespace, (b) object model store, and (c) distributed queries discovery as resource description. MOL uses extensible ad hoc scheduling policies for resource allocation. No QoS support is available, so the triggering information and breath of scope are low.

3.2.16. Legion

Legion is a software infrastructure that aims to connect multiple hosts ranging from PCs to massive parallel computers [63]. The most important features that motivate the use of legion include: (a) site autonomy, (b) support for heterogeneity, (c) usability, (d) parallel processing to achieve high system performance, (e) extensibility, (f) fault tolerance, (g) scalability, (h) security, (i) multi-language implementation support, and (j) global naming [64]. Legion appears as a vertical system and follows a hierarchical scheduling model. Legion uses distributed queries for resource discovery and periodic pull for dissemination. Moreover, Legion uses extensible structured scheduling policy for resource allocation. One of the main objectives of the Legion is the system scalability and high performance. The breadth of scope and triggering information are high in Legion.

3.2.17. Wren

Wren is a topology-based steering approach for providing network measurement [65]. The network ranges from clusters to WANs and uses information about the possible bottlenecks that may occur in the networks from topologies. The information is useful in steering the measurement techniques to calculate the channels where the bottlenecks may occur. Passive and active measurement systems are combined by Wren to minimize the measurement load [65]. Topology-based steering is used to achieve the load measurement task. Moreover, no mechanism for initial scheduling is available and a fixed application policy is used for resource allocation. Furthermore, Wren has limited scalability that results in low breadth of scope. No QoS attributes are considered in WREN and, the triggering information is also low.

3.2.18. Globus

The Globus system achieves a vertically integrated treatment of applications, networks, and middleware [66]. The low level toolkit performs: (a) communication, (b) authentication, and (c) access. Meta computing systems has the problem of configuration and performance optimization.

3.3. Cloud computing systems

Numerous cloud approaches tackle complex resource provisioning and programming problems for the users with different priorities and requirements. Eight examples of cloud computing solutions are summarized and characterized under the key system features in Table 4.

Table 5
Classification of grid, cloud and cluster systems into software and hybrid/hardware approaches.

Software only systems	Hybrid and hardware only systems
Cluster systems	
OpenMosix, Kerrighed, Gluster, Cluster-On-Demand, Enhanced MOSIX, Libra, Faucets, Nimrod/G, Tycoon, DQS, PVM, LoadLeveler, SLURM, PBS, LSF, GNQS	OpenSSI
Grid systems	
G-QoSM, 2 K, Bond, Globus, Javelin, Legion, Netsolve, Nimrod/G, Ninja, PUNCH, MOL, AppLeS, Condor, Workflow Based Approach, Grid Harvest Service, Cactus Worm, Network Weather Service	GRACE, Ninf
Cloud systems	
OpenStack, Eucalyptus	Amazon EC2, Sun Grid, Google App Engine, GRIDS Lab Aneka, Microsoft Live Mesh, GENI, E-learning ecosystem

3.3.1. Amazon Elastic Compute Cloud (EC2)

EC2 [117] is a virtual computing environment that enables a user to use web service interfaces to launch instances with a variety of operating systems. EC2 provides a rental service of VM on the Internet [68]. Amazon's EC2 service has become a standard-bearer for IaaS providers and provides many different service levels to the users [69]. Depending on the individual user choices, a new 'Machine Image' based on: (a) application types, (b) structures, (c) libraries, (d) data, and (e) associated configuration settings can be specified. The user can also choose the available Amazon Machine Images (AMIs) in the network and upload AMI to Simple Storage Service (S3). The machine can reload in a shorter period of time, for performing flexible system operations. Moreover, the whole system load time increases significantly. Virtualization is achieved by running the machines on Xen [70] at OS level. The users interact with the system through EC2 Command-line tools. EC2 is built through customizable Linux-based AMI environment.

3.3.2. Eucalyptus

Eucalyptus [71] is a Linux-based open source software framework dedicated for cloud computing. Eucalyptus allows the users to execute and control the entire VM instances deployed across a variety of physical resources. Eucalyptus is composed of a Node Controller (NC) that controls the: (a) execution, (b) inspection, and (c) termination of VM instances on the hosts running Cluster Controller (CC) [72]. CC gathers information about VM and schedules VM execution on specific NCs. Moreover, CC manages virtual instance networks. A Storage Controller (SC) called "Walrus", a storage service, provides a mechanism for storing and accessing VM images and user data [72]. Cloud Controller is the web service entry point for users and administrators that make high level scheduling decisions. Eucalyptus high-level system components are implemented as web services in a system [72]. Instance manager is responsible for virtualization in Eucalyptus. Moreover, EC2's SOAP and Query interfaces provide the system access to the user. Dynamic QoS negotiation is performed in Eucalyptus by Group Managers, who collect information through resource services.

3.3.3. Google Application Engine (GAE)

GAE [73] is a freeware platform designed for the execution of web applications. The applications are managed through a web-based administration console [74]. GAE is implemented using Java and Python. GAE provide users with a facility of authorization and authentication as a web service that lifts burden from the developers. Other than supporting Python standard library and Java, GAE also supports APIs for (a) Datastore, (b) Google accounts, (c) URL fetch, (d) image manipulation, and (e) Email services.

3.3.4. Global Environment for Network Innovations (GENI)

GENI provides a mutual and grouping environment for academia, industry, and public to catalyze revolutionary discoveries and innovation in the emerging field of global networks [75]. The project is sponsored by the National Science Foundation and is open source and broadly inclusive. GENI is a "virtual laboratory" for exploring future internets at scale [76]. The virtualization is achieved through network accessible APIs. GENI creates major opportunities to: (a) understand, (b) innovate, (c) transform global networks, and (d) interactions with society. GENI enables researchers to play with different network structures by running experimental systems within private isolated slices of a shared test bed [76]. The user can interact with the GENI interface through Slice Federation Architecture 2.0 [128]. Dynamic QoS Negotiation is also incorporated through clearing house based resource allocation. GENI can be implemented in: (a) SFA (PlanetLab), (b) ProtoGENI, and (c) GCF based environment.

3.3.5. Microsoft Live Mesh

Microsoft Live Mesh aims to provide remote access to applications and data that are stored online. The user can access the uploaded applications and data through web-based live desktop or Live Mesh software [111]. The Live Mesh software uses Windows Live Login for password-protection and is authenticated when all file transfers are protected using Secure Socket

Layers (SSLs) [111]. The concept of virtualization is implemented at the OS level. Any machine having live mesh installed can access Microsoft Live Mesh or Web-based Live Desktop.

3.3.6. Sun Network.Com (Sun Grid)

Sun Grid [78,79] is used to execute Java, C, C++, and FORTRAN based applications on the cloud. For running an application on Sun Grid, the user has to follow a certain sequence of steps. First, the user has to build and debug the applications and scripts at a local development environment. The environment configuration must be similar to that on the Sun Grid [79]. Secondly, a bundled zip archive (containing all the related scripts, libraries, executable binaries, and input data) must be created and then uploaded to Sun Grid. The virtualization is achieved through a job management system commonly termed as Sun Grid Engine. Lastly, the Sun Grid web portal or API can be used to execute and monitor the application. After the completion of application execution, the results can be downloaded to the local development environment for viewing [78,79].

3.3.7. E-learning ecosystem

E-learning ecosystem is a cloud computing based infrastructure used for the specification of all components needed for the implementation of e-learning solutions [80–82]. A fully developed e-learning ecosystem may include: (a) web-based portal, (b) access learning program, and (c) personal career aspirations. The purpose is to facilitate the users or employees to: (a) check the benefits, (b) make changes to medical plans, and (c) learn competencies that tie to the business objectives [81]. The focus of an e-learning ecosystem is to provide an infrastructure that applies business discipline to manage the learning assets and activity of the entire enterprise. The virtualization is implemented at the infrastructure layer [82]. Web based dynamic interfaces are used to interact with the users. Moreover, some value added services are also provided on demand to exclusive users.

3.3.8. Grids Lab Aneka

GRIDS Lab Aneka [83] is a service oriented architecture used in enterprise grids. The aim of Aneka is to provide a development of dynamic communication protocols that may change the preferred selection at any time. Grids Lab Aneka supports multiple application models, persistence, and security solutions [83,84]. Virtualization is an integral part and is achieved in Aneka through the resource manager and scheduler. The dynamic QoS negotiation mechanism is specified based on the SLA resource requirements. Moreover, Aneka addresses deadline (maximum time period that application needs to be completed in) and budget (maximum cost that the user is willing to pay for meeting the deadline) constraints. The user access is provided by using a workbench or a web-based portal along with value added services.

3.3.9. OpenStack

OpenStack is a large-scale open source software maintained by the collaboration of programmers for producing an open standard operating system that runs clouds for virtual computing or storage for both public and private clouds. OpenStack is composed of three software projects: (a) OpenStack Compute, (b) OpenStack Object Storage, and (c) OpenStack Image Service [113]. OpenStack Compute produces a redundant and scalable cloud computing platform by provisioning and managing large networks of VM. OpenStack Object Storage is a long-term storage system that stores multi petabytes of accessible data. OpenStack Image Service is a standard REST interface for querying information about virtual disk images. OpenStack is an open industry standard with massively scalable public cloud. Moreover, OpenStack avoids proprietary vendor lock-in by supporting all available Hypervisors abide by Apache 2.0 licensing.

4. Classification of systems

The classification of computing models (cluster, grid, and cloud) under (a) software only and (b) hardware or hybrid only systems, is discussed in the following section and is shown in Table 5. The software only classification is composed of tools, mechanisms, and policies. The hardware and hybrid classification is comprised of infrastructures or hardware oriented solutions. Any change in the hardware design and OS extensions is done by the manufacturers. The hardware and OS support can be cost prohibitive to the end-users. However, programming in the case of the addition of new hardware and software can result in more time and computational cost used. Moreover, the programming can become a big burden to end-users. The cost to change hardware and software at the user level is the least amongst all the costs associated with the system.

4.1. Software only solutions

The software only solutions are the projects that are distributed as software products, components of a software package, or as a middleware. The controlling mechanism or job schedulers are the features that are important in software only solutions. As an example of such systems, DQS and GNQS cluster queuing systems can be considered. Moreover, the crucial component is the queue management module. The other examples include OSCAR and CONDOR grid software packages. For many grid approaches, the middleware layer is a crucial layer [58]. In fact, for research purposes, the grid system can be reduced just to software only layer. Therefore, most of the grid systems presented in Table 5 is categorized as the pure software solutions.

4.2. Hardware/hybrid only solutions

Hardware/hybrid class of HPC systems is usually referred to as the multi-level cloud systems. Because the clouds are used for business purposes, strict integration of the service software application is needed with the physical devices. Moreover, the intelligent software packages are specially designed and dedicated.

5. Conclusions

The survey provides a detailed comparison and description of the three broad categories of HPC systems namely cluster, grid, and cloud. The said categories have been investigated and analyzed in terms of resource allocation. Moreover, the well-known projects and applications from each category are briefly discussed and highlighted. Furthermore, the aforementioned projects are compared on the basis of selected common features belonging to the same category. For each category, more specific characteristics are discussed. The features list can be expanded further for cluster, grid, and cloud. However, because the scope of the paper was on resource allocation only, the selected characteristics allow more clear distinctions at each level of the classification. The survey will help the readers analyze the gap between what is already available in existing systems and what is still required, so that outstanding research issues can be identified. Moreover, the features of cluster, grid, and cloud are closely related to each other and the survey will help to understand the differences. Furthermore, the systems of each category have been classified under software only and hybrid or hardware only. The hardware and OS support could be cost prohibitive to end-users. However, programming level is a big burden to end-users. Amongst the three HPC categories, grid and cloud computing appears promising and a lot of research has been conducted in each category. The focus of future HPC systems is to reduce the operational cost of data centers and increase the resilience to failure, adaptability, and graceful recovery. This survey can help new researchers to address the open areas in the research. Moreover, the survey provides the basic information along with the description of the projects in the broad domain of cluster, grid, and cloud.

Acknowledgments

The authors are thankful to Kashif Bilal and Osman Khalid for the valuable reviews, suggestions, and comments.

References

[1] G. Andrews, Foundations of Multithreaded, Parallel, and Distributed Programming, Addison–Wesley, Boston, MA, USA, 2000.
[2] Y. Amir, B. Awerbuch, A. Barak, R. Borgstrom, A. Keren, An opportunity cost approach for job assignment in a scalable computing cluster, IEEE Transactions on Parallel and Distributed Systems 11 (7) (2000) 760–768.
[3] C. Yeo, R. Buyya, A taxonomy of market-based resource management systems for utility-driven cluster computing, Software: Practice and Experience 36 (13) (2006) 1381–1419.
[4] D. Irwin, L. Grit, J. Chas, Balancing risk and reward in a market-based task service, in: 13th International Symposium on High Performance, Distributed Computing (HPDC13), June 2004, pp. 160–169.
[5] C. Yeo, R. Buyya, Service level agreement based allocation of cluster resources: handling penalty to enhance utility, in: 7th IEEE International Conference on Cluster Computing (Cluster 2005), September 2005.
[6] J. Sherwani, N. Ali, N. Lotia, Z. Hayat, R. Buyya, Libra: a computational economy-based job scheduling system for clusters, Software: Practice and Experience 34 (6) (2004) 573–590.
[7] T. Casavant, J. Kuhl, A taxonomy of scheduling in general-purpose distributed computing systems, IEEE Transactions on Software Engineering 14 (2) (1988) 141–154.
[8] J. Regeh, J. Stankovic, M. Humphrey, The case for hierarchical schedulers with performance guarantees, TR-CS 2000-07, Department of Computer Science, University of Virginia, March 2000, p. 9.
[9] R. Wolski, N. Spring, J. Hayes, Predicting the CPU availability of time-shared Unix systems on the computational grid, in: Proceedings of the 8th High-Performance Distributed Computing Conference, August 1999.
[10] E. Huedo, R. Montero, I. Llorente, A framework for adaptive execution in grids, Software-Practice and Experience 34 (07) (2004) 631–651.
[11] D. Benslimane, D. Schahram, S. Amit, Services Mashups: the new generation of web applications, IEEE Internet Computing 12 (5) (2008) 13–15.
[12] C. Diaz, M. Guzek, J. Pecero, P. Bouvry, S. Khan, Scalable and energy-efficient scheduling techniques for large-scale systems, in: 11th IEEE International Conference on Computer and Information Technology (CIT), September 2011, pp. 641–647.
[13] A. Barak, O. La'adan, The MOSIX multicomputer operating system for high performance cluster computing, Future Generation Computer Systems 13 (4–5) (March 1998) 361–372.
[14] Gluster, www.gluster.org, accessed February 16, 2012.
[15] P. Lindberg, J. Leingang, D. Lysaker, K. Bilal, S. U. Khan, P. Bouvry, N. Ghani, N. Min-Allah, J. Li, Comparison and analysis of greedy energy-efficient scheduling algorithms for computational grids, in: A. Y. Zomaya, Y.-C. Lee (Eds.), Energy Aware Distributed Computing Systems, John Wiley & Sons, Hoboken, NJ, USA, 2012, ISBN 978-0-470-90875-4, Chapter 7.
[16] M. Bhandarkar, L. Kal'e, E. Sturler, J. Hoeflinger, Adaptive load balancing for MPI programs, Lecture Notes in Computer Science (LNCS) 2074 (2001) 108–117.
[17] G. L. Valentini, S. U. Khan, P. Bouvry, Energy-efficient resource utilization in cloud computing, in: A. Y. Zomaya, H. Sarbazi-Azad (Eds.), Large Scale Network-centric Computing Systems, John Wiley & Sons, Hoboken, NJ, USA, 2013, ISBN: 978-0-470-93688-7, Chapter 16.
[18] L. Kal'e, S. Kumar, J. DeSouza, A malleable-job system for timeshared parallel machines, in: 2nd International Symposium on Cluster Computing and the Grid (CCGrid 2002), May 2002, pp. 215–222.
[19] DQS, http://www.msi.umn.edu/sdvl/info/dqs/dqs-intro.html, accessed March 20, 2011.
[20] K. Lai, L. Rasmusson, E. Adar, L. Zhang, B. Huberman, Tycoon: an implementation of a distributed, market-based resource allocation system, Multiagent Grid System 1 (3) (2005) 169–182.
[21] A. Bernardo, H. Lai, L. Fine, Tycoon: a distributed market-based resource allocation system, TR-arXiv:cs.DC/0404013, HP Labs, Palo Alto, CA, USA, February 2008, p. 8.
[22] J. Chase, D. Irwin, L. Grit, J. Moore, S. Sprenkle, Dynamic virtual clusters in a grid site manager, in: 12th International Symposium on High Performance, Distributed Computing (HPDC12), June 2003, pp. 90–100.

[23] C. Morin, R. Lottiaux, G. Vallee, P. Gallard, G. Utard, R. Badrinath, L. Rilling, Kerrighed: a single system image cluster operating system for high performance computing, in: Proceedings of Europar 2003 Parallel Processing, Lecture Notes in Computer Science, vol. 2790, August 2003, pp. 1291–1294.
[24] Kerrighed, http://kerrighed.org/wiki/index.php/Main_Page, accessed March 15, 2011.
[25] OpenSSI, http://openssi.org/cgi-bin/view?page=openssi.html, accessed March 12, 2011.
[26] J. Sherwani, N. Ali, N. Lotia, Z. Hayat, R. Buyya, Libra: a computational economy based job scheduling system for clusters, International Journal of Software: Practice and Experience 34 (06) (2004) 573–590.
[27] C. Yeo, R. Buyya, Pricing for utility-driven resource management and allocation in clusters, International Journal of High Performance Computing Applications 21 (4) (2007) 405–418.
[28] P. Springer, PVM support for clusters, in: 3rd IEEE International Conference on, Cluster Computing (CLUSTER'01), October 2001.
[29] B. Cooper, H. Garcia-Molina, Bidding for storage space in a peer-to-peer data preservation system, in: 22nd International Conference on Distributed Computing Systems (ICDCS 2002), July 2002, pp. 372–381.
[30] B. Chun, D. Culler, Market-based proportional resource sharing for clusters, TR-CSD-1092, Computer Science Division, University of California, Berkeley, USA, January 2000, p. 19.
[31] GNQS, http://gnqs.sourceforge.net/docs/starter_pack/introducing/index.html, accessed January 20, 2011.
[32] Workload management with load leveler, http://www.redbooks.ibm.com/abstracts/sg246038.html, accessed February 10, 2011.
[33] R. Buyya, D. Abramson, J. Giddy, A case for economy grid architecture for service oriented grid computing, in: 15th International Parallel and Distributed Processing Symposium, April 2001, pp. 776–790.
[34] D. Batista, N. Fonseca, A brief survey on resource allocation in service oriented grids, in: IEEE Globecom Workshops, November 2007, pp. 1–5.
[35] H. Nakada, M. Sato, S. Sekiguchi, Design and implementation of Ninf: towards a global computing infrastructure, Future Generation Computing Systems 15 (5–6) (1999) 649–658 (Meta-computing Special Issue).
[36] K. Krauter, R. Buyya, M. Maheswaran, A taxonomy and survey of grid resource management systems for distributed computing, Journal of Software Practice and Experience 32 (2) (2002) 135–164, http://dx.doi.org/10.1002/spe.432.
[37] R. Al-Ali, A. Hafid, O. Rana, D. Walker, QoS adaptation in service-oriented grids, Performance Evaluation 64 (7–8) (2007) 646–663.
[38] R. Al-Ali, O. Rana, D. Walker, S. Jha, S. Sohail, G-QoSM: grid service discovery using QoS properties, Computing and Informatics Journal 21 (4) (2002) 363–382 (Special Issue on Grid Computing).
[39] F. Pinel, J. Pecero, S. Khan, P. Bouvry, A review on task performance prediction in multi-core based systems, in: 11th IEEE International Conference on Computer and Information Technology (CIT), September 2011, pp. 615–620.
[40] M. Neary, A. Phipps, S. Richman, P. Cappello, Javelin 2.0: Java-based parallel computing on the internet, in: European Parallel Computing Conference (Euro-Par 2000), August 2000, pp. 1231–1238.
[41] X. Sun, M. Wu, GHS: a performance system of grid computing, in: 19th IEEE International Parallel and Distributed Processing Symposium, April 2005.
[42] J. Blythe, S. Jain, E. Deelman, Y. Gil, K. Vahi, A. Mandal, K. Kennedy, Task scheduling strategies for workflow-based applications in grids, in: IEEE International Symposium on Cluster Computing and Grids (CCGRID'05), vol. 2, May 2005, pp. 759–767.
[43] D. Carvalho, F. Kon, F. Ballesteros, M. Romn, R. Campbell, D. Mickunas, Management of execution environments in 2K, in: 7th International Conference on Parallel and Distributed Systems (ICPADS '00), July 2000, pp. 479–485.
[44] F. Kon, R. Campbell, M. Mickunas, K. Nahrstedt, 2K: a distributed operation system for dynamic heterogeneous environments, in: 9th IEEE International Symposium on High Performance Distributed Computing (HPDC '00), August 2000, pp. 201–210.
[45] M. Roman, F. Kon, R. H. Campbell, Design and implementation of runtime reflection in communication middleware the dynamic use case, in: Workshop on Middleware (ICDCS'99), May 1999.
[46] D. Schmidt, Distributed object computing with CORBA Middleware, http://www.cs.wustl.edu/~schmidt/corba.html, accessed February 4, 2011.
[47] R. Wolski, N. Spring, J. Hayes, The network weather service: a distributed resource performance forecasting service for metacomputing, Future Generation Computer Systems 15 (5) (1999) 757–768.
[48] F. Berman, R. Wolski, The AppLeS project: a status report, in: 8th NEC Research Symposium, May 1997.
[49] P. Chandra, A. Fisher, C. Kosak, Ng. TSE, P. Steenkiste, E. Takahashi, H. Zhang, Darwin: customizable resource management for value-added network services, in: 6th IEEE International Conference on Network Protocols, October 1998.
[50] M. Litzkow, M. Livny, M. Mutka, Condor – a hunter of idle workstations, in: 8th International Conference of Distributed Computing Systems, June 1988, pp. 104–111.
[51] J. Basney, M. Livny, Deploying a high throughput computing cluster, in: R. Buyya (Ed.), High Performance Cluster Computing, vol. 1, Prentice Hall, 1999, pp. 116–134.
[52] G. Allen, D. Angulo, I. Foster, G. Lanfermann, C. Liu, T. Radke, E. Seidel, J. Shalf, The cactus worm: experiments with dynamic resource discovery and allocation in a grid environment, International Journal of High Performance Computing Applications 15 (4) (2001) 345–358.
[53] F. Berman, A. Chien, K. Cooper, J. Dongarra, I. Foster, D. Gannon, L. Johnsson, K. Kennedy, C. Kesselman, J. Mellor-Crummey, D. Reed, L. Torczon, R. Wolski, The GrADS project: software support for high-level grid application development, International Journal of Supercomputer Applications 15 (04) (2001) 327–344.
[54] N. Kapadia, J. Fortes, PUNCH: an architecture for web-enabled wide-area network-computing, The Journal of Networks, Software Tools and Applications 2 (2) (1999) 153–164 (Special Issue on High Performance Distributed Computing).
[55] L. Wang, S.U. Khan, Review of performance metrics for green data centers: a taxonomy study, Journal of Supercomputing 63 (3) (2013) 639–656.
[56] D. Abramson, J. Giddy, L. Kotler, High performance parametric modeling with Nimrod/G: killer application for the global grid?, in: International Parallel and Distributed Processing Symposium (IPDPS 2000), May 2000, pp. 520–528.
[57] R. Buyya, D. Abramson, J. Giddy, Nimrod/G: an architecture for a resource management and scheduling system in a global computational grid, in: International Conference on High Performance Computing in Asia–Pacific Region (HPC Asia 2000), vol. 1, May 2000, pp. 283–289.
[58] R. Buyya, J. Giddy, D. Abramson, An evaluation of economy-based resource trading and scheduling on computational power grids for parameter sweep applications, in: 2nd International Workshop on Active Middleware Services (AMS'00), August 2000.
[59] L. Kotler, D. Abramson, P. Roe, D. Mather, Activesheets: super-computing with spreadsheets, in: Advanced Simulation Technologies Conference High Performance Computing, Symposium (HPC'01), April 2001. .
[60] H. Casanova, J. Dongarra, Netsolve: a network-enabled server for solving computational science problems, International Journal of Supercomputer Applications and High Performance Computing 11 (3) (1997) 212–223.
[61] J. Gehring, A. Streit, Robust resource management for metacomputers, in: 9th IEEE International Symposium on High Performance Distributed Computing, August 2000.
[62] I. Foster, C. Kesselman, Globus: a metacomputing infrastructure toolkit, International Journal of Supercomputer Applications 11 (2) (1996) 115–128.
[63] S. Chapin, J. Karpovich, A. Grimshaw, The legion resource management system, in: 5th Workshop on Job Scheduling Strategies for Parallel Processing, April 1999, pp. 162–178.
[64] S. Andrew, W. Wulf, The legion vision of a worldwide virtual computer, Communications of ACM 40 (1) (1997) 39–45.
[65] B. Lowekamp, Combining active and passive network measurements to build scalable monitoring systems on the grid, ACM SIGMETRICS Performance Evaluation Review 30 (4) (2003) 19–26.
[66] G. Valentini, W. Lassonde, S. Khan, N. Min-Allah, S. Madani, J. Li, L. Zhang, L. Wang, N. Ghani, J. Kolodziej, H. Li, A. Zomaya, C. Xu, P. Balaji, A. Vishnu, F. Pinel, J. Pecero, D. Kliazovich, P. Bouvry, An overview of energy efficiency techniques in cluster computing systems, Cluster Computing 16 (1) (2011) 1–13, http://dx.doi.org/10.1007/s10586-011-0171-x.

[67] S. Khan, I. Ahmad, A cooperative game theoretical technique for joint optimization of energy consumption and response time in computational grids, IEEE Transactions on Parallel and Distributed Systems 20 (3) (2009) 346–360.
[68] S. Toyoshima, S. Yamaguchi, M. Oguchi, Storage access optimization with virtual machine migration and basic performance analysis of Amazon EC2, in: IEEE 24th International Conference on Advanced Information Networking and Applications Workshops, Apr. 2010, pp. 905–910.
[69] Z. Hill, M. Humphrey, A quantitative analysis of high performance computing with Amazon's EC2 infrastructure: the death of the local cluster?, in: 10th IEEE/ACM International Conference on Grid Computing, October 2009, pp. 26–33.
[70] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, A. Warfield, Xen and the art of virtualization, ACM Symposium on Operating Systems Principles (SOSP) 37 (5) (2003) 164–177.
[71] D. Nurmi, R. Wolski, C. Grzegorczyk, G. Obertelli, S. Soman, L. Youseff, D. Zagorodnov, The eucalyptus open-source cloud computing system, in: 9th IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGRID '09), May 2009, pp. 124–31.
[72] F. Lombardi, R. DiPietro, Secure virtualization for cloud computing, Journal of Network and Computer Application 34 (4) (2011) 1113–1122.
[73] A. Bedra, Getting started with Google app engine and Clojure, IEEE Journal of Internet Computing 14 (4) (2010) 85–88.
[74] Google App Engine, http://appengine.google.com, accessed February 02, 2011.
[75] GENI, http://www.geni.net, acessed February 02, 2011.
[76] I. Baldine, Y. Xin, A. Mandal, C. Heermann, Networked cloud orchestration: a GENI perspective, in: IEEE GLOBECOM Workshops, December 2010, pp. 573–578.
[77] J. Kolodziej, S.U. Khan, Data scheduling in data grids and data centers: a short taxonomy of problems and intelligent resolution techniques, Transactions on Computational Collective Intelligence, vol. X (2013) pp. 103–119.
[78] Sun Network.com (Sun Grid), http://www.network.com, accessed February 08, 2011.
[79] W. Gentzsch, Sun grid engine: towards creating a compute power grid, in: 1st IEEE/ACM International Symposium on Cluster Computing and the Grid, May 2001, pp. 35–36.
[80] L. Uden, E. Damiani, The Future of E-learning: E-learning ecosystem, in: 1st IEEE International Conference on Digital Ecosystems and Technologies, June 2007, pp. 113–117.
[81] V. Chang, C. Guetl, E-Learning ecosystem (ELES)-a holistic approach for the development of more effective learning environment for small-and-medium sized enterprises, in: 1st IEEE International Conference on Digital Ecosystems and Technologies, February 2007, pp. 420–425.
[82] B. Dong, Q. Zheng, J. Yang, H. Li, M. Qiao, An e-learning ecosystem based on cloud computing infrastructure, in: 9th IEEE International Conference on Advanced Learning Technologies, July 2009, pp. 125–127.
[83] X. Chu, K. Nadiminti, C. Jin, S. Venugopal, R. Buyya, Aneka: next-generation enterprise grid platform for e-science and e-business applications, in: 3rd IEEE International Conference on e-Science and Grid Computing, December 2007, pp. 151–159.
[84] A. Chien, B. Calder, S. Elbert, K. Bhatia, Entropia: architecture and performance of an enterprise desktop grid system, Journal of Parallel and Distributed Computing 63 (5) (2003) 597–610.
[85] P. Kokkinos, E. Varvarigos, A framework for providing hard delay guarantees and user fairness in grid computing, Future Generation Computer Systems 25 (6) (2009) 674–686.
[86] E. Varvarigos, Routing and scheduling in grids, in: 10th Anniversary International Conference on Transport and Optical Networks, June 2008, pp. 170–174.
[87] P. Berstein, http://research.microsoft.com/en-us/people/philbe/chapter3.pdf, accessed July 25, 2011.
[88] P. Wieder, O. Waldrich, W. Ziegler, Advanced techniques for scheduling, reservation and access management for remote laboratories and instruments, in: 2nd IEEE International Conference on e-Science and Grid, Computing (e-Science'06), December 2006, pp. 128–128.
[89] Vmware, Inc., http://www.vmware.com/pdf/vsphere4/r41/vsp_41_resource_mgmt.pdf, accessed July 23, 2011.
[90] http://www.platform.com/workload-management/high-performance-computing, accessed August 07, 2011.
[91] M. Jette, A. Yoo, M. Grondona, SLURM: simple linux utility for resource management, D.G. Feitelson, L. Rudolph (Eds.), Job Scheduling Strategies for Parallel Processing, 2003, pp. 37–51.
[92] Research computing and cyber infrastructure, http://rcc.its.psu.edu/user_guides/system_utilities/pbs/, accessed August 07, 2011.
[93] G. White, M. Quartly, http://www.ibm.com/developerworks/systems/library/es-linuxclusterintro, accessed February 15, 2012.
[94] Grid computing, http://www.adarshpatil.com/newsite/images/grid-computing.gif, accessed February 20, 2012.
[95] I. Foster, C. Kesselman, S. Tuecke, The anatomy of the grid, International Journal of Supercomputer Applications 15 (3) (2001) 200–222.
[96] J. Leung, Handbook of Scheduling: Algorithms, Models, and Performance Analysis, 1st ed., CRC Press Inc., Boca Raton, FL, USA, 2004.
[97] P. Dutot, L. Eyraud, G. Mounie, D. Trystram, Bi-criteria algorithm for scheduling jobs on cluster platforms, in: 16th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA), July 2004, pp. 125–132.
[98] N. Arora, R. Blumofe, C. Plaxton, Thread scheduling for multi-programmed multi-processors, Theory of Computing Systems 34 (2) (2001) 115–144.
[99] J. Kolodziej, S. Khan, F. Xhafa, Genetic Algorithms for energy-aware scheduling in computational grids, in: 6th IEEE International Conference on P2P, Parallel, Grid, Cloud, and, Internet Computing (3PGCIC), October 2001, pp. 17–24.
[100] K. Ramamritham, J. Stankovic, Scheduling algorithm and operating system support for real-time systems, Proceedings of IEEE 82 (1) (2002) 55–67.
[101] I. Foster, Y. Zhao, I. Raicu, S. Lu, Cloud computing and grid computing 360-degree compared, in: Grid Computing Environments Workshop 2008 (GCE'08), November 2008, pp. 1–10.
[102] What is the difference between Cloud, Cluster and Grid Computing?, http://www.cloud-competence-center.de/understanding/difference-cloud-cluster-grid/, accessed February 5, 2011.
[103] T. Xie, A. Sung, X. Qin, M. Lin, L. Yang, Real-time scheduling with quality of security constraints, International Journal of High Performance Computing and Networking 4 (3) (2006) 188–197.
[104] P. Brucker, Scheduling Algorithms, 4th ed., Springer-Verlag, Guildford, Surrey, UK, 2004.
[105] F. Pinel, J. Pecero, P. Bouvry, S. Khan, A two-phase heuristic for the scheduling of independent tasks on computational grids, in: ACM/IEEE/IFIP International Conference on High Performance Computing and Simulation (HPCS), July 2011, pp. 471–477.
[106] Continuous Availability for Enterprise Messaging: Reducing Operational Risk and Administration Complexity, http://www.progress.com/docs/whitepapers/public/sonic/sonic_caa.pdf, accessed February 12, 2011.
[107] Cluster computing, http://searchdatacenter.techtarget.com/definition/cluster-computing, accessed February 03, 2011.
[108] Parallel sysplex, http://www-03.ibm.com/systems/z/advantages/pso/, accessed February 22, 2011.
[109] L. Kal'e, S. Krishnan, Charm++: parallel programming with message-driven objects, in: G.V. Wilson, P. Lu (Eds.), Parallel Programming Using C++, MIT Press, Cambridge, MA, USA, 1996, pp. 175–213.
[110] N. Kapadia, R. Figueiredo, J. Fortes, PUNCH: Web portal for running tools, IEEE Micro 20 (3) (2000) 38–47.
[111] Microsoft Live Mesh, http://www.mesh.com, accessed February 12, 2011.
[112] N. Sadashiv, S. Kumar, Cluster, grid and cloud computing: a detailed comparison, in: 6th International Conference on Computer Science & Education (ICCSE), September 2011, pp. 477–482.
[113] OpenSatck, http://openstack.org/downloads/openstack-overview-datasheet.pdf, accessed April 04, 2012.
[114] Globus Nimbus, http://www.nimbusproject.org/doc/nimbus/faq/, accessed April 05, 2012.
[115] Open Nebula, http://opennebula.org/, accessed April 05, 2012.
[116] L. Kal'e, S. Kumar, M. Potnuru, J. DeSouza, S. Bandhakavi, Faucets: efficient resource allocation on the computational grid, in: 33rd International Conference on Parallel Processing (ICPP 2004), August 2004.
[117] Amazon elastic compute cloud (EC2), http://www.amazon.com/ec2/, accessed February 10, 2011.

[118] L. Wang, J. Tao, H. Marten, A. Streit, S.U. Khan, J. Kolodziej, D. Chen, MapReduce across distributed clusters for data-intensive applications, in: IEEE 26th International Parallel and Distributed Processing Symposium Workshops & PhD Forum (IPDPSW), May 2012, pp. 2004, 2011, 21–25. http://dx.doi.org/10.1109/IPDPSW.2012.249.
[119] I. Foster, C. Kesselman, J. Nick, S. Tuecke, The physiology of the grid an open grid services architecture for distributed systems integration, Argonne National Laboratory, Mathematics and Computer Science Division Chicago, January 2002, 37 pp., www.globus.org/research/papers/ogsa.pdf.
[120] R. Buyya, R. Ranjan, R.N. Calheiros, Modeling and simulation of scalable cloud computing environments and the cloudsim toolkit: challenges and opportunities, Proceedings of the 7th High Performance Computing and Simulation Conference HPCS 2009, IEEE Press, New York, USA, Leipzig, Germany, 2009. June 21–24.
[121] K.K. Droegemeier, D. Gannon, D. Reed, B. Plale, J. Alameda, T. Baltzer, K. Brewster, R. Clark, B. Domenico, S. Graves, E. Joseph, D. Murray, R. Ramachandran, M. Ramamurthy, L. Ramakrishnan, J.A. Rushing, D. Weber, R. Wilhelmson, A. Wilson, M. Sue, S. Yalda, Service-oriented environments for dynamically interacting with mesoscale weather, Computing in Science and Engineering 7 (6) (2005) 12–29.
[122] F. Dong, S.G. Akl, Scheduling algorithms for grid computing: state of art and open problems. Queens University. Technical report. http://www.cs.queensu.ca/TechReports/Reports/2006-504.pdf.
[123] Amazon's HPC cloud: supercomputing for the 99%, http://arstechnica.com/business/2012/05/amazons-hpc-cloud-supercomputing-for-the-99/, accessed 11 Sep, 2012.
[124] L. Wang, W. Jie, J. Chen, Grid Computing: Infrastructure, Service, and Applications, CRC Press, Kindle Edition, 2009. pp. 338.
[125] F. Berman, R. Wolski, S. Figueira, J. Schopf, G. Shao, Application level scheduling on distributed heterogeneous networks, in: Proceedings of Supercomputing, 1996.
[126] N. Spring, R. Wolski, Application level scheduling: gene sequence library comparison, in: Proceedings of ACM International Conference on Supercomputing, July 1998.
[127] D. Andresen, T. McCune, Towards a hierarchical scheduling system for distributed WWW server clusters, in: Proceedings of the Seventh IEEE International Symposium on High Performance Distributed Computing (HPDC).
[128] Slice Federation Architecture 2.0, GENI, http://groups.geni.net/geni/wiki/SliceFedArch, accessed September 17, 2012.
[129] L. Skorin-Kapov, M. Matijasevic, Dynamic QoS negotiation and adaptation for networked virtual reality services, in: IEEE WoWMoM '05, Taormina, Italy, June 2005, pp. 344–351.
[130] S. Iqbal, R. Gupta, Y. Lang, Job scheduling in HPC clusters, Power Solutions (2005) 133–135.
[131] S. Senapathi, D.K. Panda, D. Stredney, H.-W. Shen, A QoS framework for clusters to support applications with resource adaptivity and predictable performance, in: Proceedings of the IEEE International Workshop on Quality of Service (IWQoS), May 2002.
[132] K.H. Yum, E.J. Kim, C. Das, QoS provisioning in clusters: an investigation of router and NIC design, in: ISCA-28, 2001.
[133] S. Ali, T.D. Braun, H.J. Siegel, A.A. Maciejewski, N. Beck, L. Boloni, M. Maheswaran, A.I. Reuther, J.P. Robertson, M.D. Theys, B. Yao, Characterizing resource allocation heuristics for heterogeneous computing systems, in: A.R. Hurson (Ed.), Advances in Computers, Parallel, Distributed, and Pervasive Computing, vol. 63, Elsevier, Amsterdam, the Netherlands, 2005, pp. 91–128.
[134] F. Pascual, K. Rzadca, D. Trystram, Cooperation in multi-organization scheduling, Concurrency and Computation: Practice and Experience 21 (7) (2009) 905–921.
[135] Y.S. Kee, D. Logothetis, R. Huang, H. Casanova, A. Chien, Efficient resource description and high quality selection for virtual grids, in: Proceedings of CCGrid, 2005.
[136] M. Tchiboukdjian, N. Gast, D. Trystram, Decentralized list scheduling, Annals of Operations Research 207 (1) (2012) 1–23.
[137] S.M. Mostafa, S.Z. Rida, S.H. Hamad, Finding time quantum of round robin CPU scheduling algorithm in general computing systems using integer programming, International Journal of Research and Reviews in Applied Sciences (IJRRAS) 5 (1) (2010) 64–71.
[138] HP Cloud, https://www.hpcloud.com/pricing, accessed August 08, 2013.
[139] D. Chen, L. Wang, X. Wu, J. Chen, S.U. Khan, J. Kolodziej, M. Tian, F. Huang, W. Liu, Hybrid modelling and simulation of huge crowd over a hierarchical grid architecture, Future Generation Computer Systems 29 (5) (2013) 1309–1317.
[140] J. Kolodziej, S.U. Khan, Multi-level hierarchical genetic-based scheduling of independent jobs in dynamic heterogeneous grid environment, Information Sciences 214 (2012) 1–19.
[141] S.U. Khan, A goal programming approach for the joint optimization of energy consumption and response time in computational grids, in: 28th IEEE International Performance Computing and Communications Conference (IPCCC), Phoenix, AZ, USA, December 2009, pp. 410–417.
[142] J. Kołodziej, S.U. Khan, L. Wang, M. Kisiel-Dorohinicki, S.A. Madani, E. Niewiadomska-Szynkiewicz, A.Y. Zomaya, C. Xu, Security, energy, and performance-aware resource allocation mechanisms for computational grids, Future Generation Computer Systems, October 2012, ISSN 0167-739X, http://dx.doi.org/10.1016/j.future.2012.09.009.
[143] G.L. Valentini, W. Lassonde, S.U. Khan, N. Min-Allah, S.A. Madani, J. Li, L. Zhang, L. Wang, N. Ghani, J. Kolodziej, H. Li, A.Y. Zomaya, C.-Z. Xu, P. Balaji, A. Vishnu, F. Pinel, J.E. Pecero, D. Kliazovich, P. Bouvry, An overview of energy efficiency techniques in cluster computing systems, Cluster Computing 16 (1) (2013) 3–15.
[144] F. Pinel, J.E. Pecero, S.U. Khan, P. Bouvry, Energy-efficient scheduling on milliclusters with performance constraints, in: ACM/IEEE International Conference on Green Computing and Communications (GreenCom), Chengdu, Sichuan, China, August 2011, pp. 44–49.
[145] L. Wang, S.U. Khan, D. Chen, J. Kolodziej, R. Ranjan, C.-Z. Xu, A.Y. Zomaya, Energy-aware parallel task scheduling in a cluster, Future Generation Computer Systems 29 (7) (2013) 1661–1670.
[146] L. Wang, J. Tao, H. Marten, A. Streit, S.U. Khan, J. Kolodziej, D. Chen, MapReduce across distributed clusters for data-intensive applications, in: 26th IEEE International Parallel and Distributed Processing Symposium (IPDPS), Shanghai, China, May 2012, pp. 2004–2011.
[147] J. Kolodiej, S.U. Khan, E. Gelenbe, E.-G. Talbi, Scalable optimization in grid, cloud, and intelligent network computing, Concurrency and Computation: Practice and Experience 25 (12) (2013) 1719–1721.
[148] D. Kliazovich, P. Bouvry, Y. Audzevich, S.U. Khan, GreenCloud: a packet-level simulator of energy-aware cloud computing data centers, in: 53rd IEEE Global Communications Conference (Globecom), Miami, FL, USA, December 2010.
[149] K. Bilal, S.U. Khan, L. Zhang, H. Li, K. Hayat, S.A. Madani, N. Min-Allah, L. Wang, D. Chen, M. Iqbal, C.-Z. Xu, A.Y. Zomaya, Quantitative comparisons of the state of the art data center architectures, Concurrency and Computation: Practice and Experience 25 (12) (2013) 1771–1783.
[150] J. Shuja, S.A. Madani, K. Bilal, K. Hayat, S.U. Khan, S. Sarwar, Energy-efficient data centers, Computing 94 (12) (2012) 973–994.