SOFTWARE—PRACTICE AND EXPERIENCE
Softw. Pract. Exper. 2006; **36**:1381–1419
Published online 8 June 2006 in Wiley InterScience (www.interscience.wiley.com). DOI: 10.1002/spe.725

A taxonomy of market-based resource management systems for utility-driven cluster computing

Chee Shin Yeo and Rajkumar Buyya*,†

Grid Computing and Distributed Systems Laboratory,
Department of Computer Science and Software Engineering, The University of Melbourne, Australia

SUMMARY

In utility-driven cluster computing, cluster Resource Management Systems (RMSs) need to know the specific needs of different users in order to allocate resources according to their needs. This in turn is vital to achieve service-oriented Grid computing that harnesses resources distributed worldwide based on users' objectives. Recently, numerous market-based RMSs have been proposed to make use of real-world market concepts and behavior to assign resources to users for various computing platforms. The aim of this paper is to develop a taxonomy that characterizes and classifies how market-based RMSs can support utility-driven cluster computing in practice. The taxonomy is then mapped to existing market-based RMSs designed for both cluster and other computing platforms to survey current research developments and identify outstanding issues. Copyright © 2006 John Wiley & Sons, Ltd.

Received 9 December 2004; Revised 4 August 2005; Accepted 27 September 2005

KEY WORDS: cluster computing; utility computing; market-based resource management systems; taxonomy

1. INTRODUCTION

Next-generation scientific research involves solving Grand Challenge Applications (GCAs) that demand ever increasing amounts of computing power. Recently, a new type of High-Performance Computing (HPC) paradigm called *cluster computing* [1–3] has become a more viable choice for executing these GCAs since cluster systems are able to offer equally high-performance with a lower price compared with traditional supercomputing systems. A cluster system comprises of independent machines that are connected by high-speed networks and uses middlewares to create an illusion of

*Correspondence to: Rajkumar Buyya, Department of Computer Science and Software Engineering, The University of Melbourne, VIC 3010, Australia.
†E-mail: raj@cs.mu.oz.au

a single system [4] and hide the complexities of the underlying cluster architecture from the users. For example, the *cluster Resource Management System (RMS)* provides a uniform interface for user-level sequential and parallel applications to be executed on the cluster system and thus hides the existence of multiple cluster nodes from users.

The cluster RMS supports four main functionalities: resource management; job queuing; job scheduling; and job execution. It manages and maintains status information of the resources such as processors and disk storage in the cluster system. Jobs submitted into the cluster system are initially placed into queues until there are available resources to execute the jobs. The cluster RMS then invokes a scheduler to determine how resources are assigned to jobs. After that, the cluster RMS dispatches the jobs to the assigned nodes and manages the job execution processes before returning the results to the users upon job completion.

In cluster computing, the *producer* is the owner of the cluster system that provides resources to accomplish users' service requests. Examples of resources that can be utilized in a cluster system are processor power, memory storage and data storage. The *consumer* is the user of the resources provided by the cluster system and can be either a physical human user or a software agent that represents a human user and acts on his behalf. A cluster system has multiple consumers submitting job requests that need to be executed.

In *utility-driven cluster computing*, consumers have different requirements and needs for various jobs and thus can assign value or utility to their job requests. During job submission to the cluster RMS, consumers can specify their requirements and preferences for each respective job using Quality of Service (QoS) parameters. The cluster RMS then considers these QoS parameters when making resource allocation decisions. This provides a user-centric approach with better user personalization since consumers can potentially affect the resource allocation outcomes, based on their assigned utility. Thus, the objective of the cluster RMS is to maximize overall consumers' utility satisfaction. For example, the cluster RMS can achieve this objective from either the job perspective, where it maximizes the number of jobs whose QoS is satisfied or the consumer perspective, where it maximizes the aggregate utility perceived by individual consumers.

Existing cluster RMSs such as Condor [5], LoadLeveler [6], Load Sharing Facility (LSF) [7], Portable Batch System (PBS) [8] and Sun Grid Engine (SGE) [9] are not viable to support utility-driven cluster computing since they still adopt system-centric resource allocation approaches that focus on optimizing overall cluster performance. For example, these cluster RMSs aim to maximize processor throughput and utilization for the cluster and minimize the average waiting time and response time for the jobs. These system-centric approaches assume that all job requests are of equal user importance and thus neglect actual levels of service required by different users. They do not employ utility models for allocation and management of resources that would otherwise consider and thus achieve the desired utility for cluster users and owners. Therefore, these existing cluster RMSs need to be extended to support utility-driven cluster computing.

The advent of Grid computing [10] further reinforces the necessity for utility-driven cluster computing. In service-oriented Grid computing [11], users can specify various levels of service required for processing their jobs on a Grid. Grid schedulers such as Grid brokers [12,13] and Grid workflow engines [14] then make use of this user-specific information to discover available Grid resources and determine the most suitable Grid resource to submit the jobs to. Currently, cluster systems dominate the majority of Grid Resources whereby Grid schedulers submit and monitor their jobs being executed on the cluster systems through interaction with their cluster RMS.

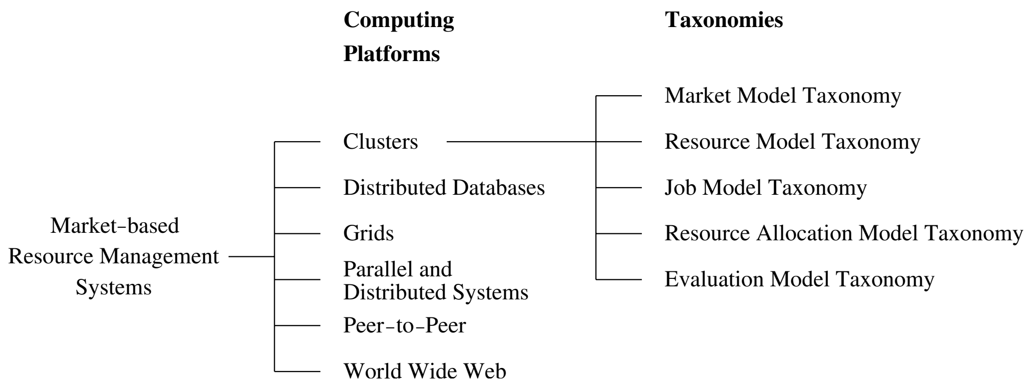

Figure 1. Categorization of market-based RMSs.

Examples of large-scale Grid systems that are composed of cluster systems includes the TeraGrid [15] in the United States, LHC Computing Grid [16] in Europe, NAREGI [17] in Japan, and APAC Grid [18] in Australia.

In addition, commercial vendors are progressing aggressively towards providing a service market through Grid computing. For instance, IBM's E-Business On Demand [19], HP's Adaptive Enterprise [20] and Sun Microsystem's pay-as-you-go [21] are using Grid technologies to provide dynamic service delivery where users only pay for what they use and thus save from investing heavily on computing facilities. Vendors and respective users have to agree on Service Level Agreements (SLAs) that serve as contracts outlining the expected level of service performance such that vendors are liable to compensate users for any service under-performance. Cluster RMSs thus need to support SLA-based resource allocations that not only balance competing user needs but also enhance the profitability of the cluster owner while delivering the expected level of service performance. This reinforces the significance of using market-based mechanisms to enable utility-driven cluster computing. Market concepts and mechanisms incorporated at the cluster computing level can enforce SLAs to deliver utility and facilitate easy extensions to support Grid economy [22] for service-oriented Grids.

Market-based RMSs have been utilized in many different computing platforms: clusters [23–25]; distributed databases [26,27]; Grids [28–30]; parallel and distributed systems [31–33]; peer-to-peer [34]; and World Wide Web [35–37] (see Figure 1). They have a greater emphasis on user QoS requirements as opposed to traditional RMSs that focus on maximizing system usage. Market concepts can be used to prioritize competing jobs and assign resources to jobs according to users' valuations for QoS requirements and cluster resources.

Market-based cluster RMSs need to support three requirements in order to enable utility-driven cluster computing [23]: (i) provide a means for users to specify their QoS needs and valuations; (ii) utilize policies to translate the valuations into resource allocations; and (iii) support mechanisms to enforce the resource allocations in order to achieve each individual user's perceived value or utility. The first requirement allows the market-based cluster RMS to be aware of user-centric service requirements so that competing service requests can be prioritized more accurately. The second requirement then determines how the cluster RMS can allocate resources appropriately and effectively to different requests by considering the solicited service requirements. The third requirement finally needs the underlying cluster operating system mechanisms to recognize and enforce the assigned resource allocations.

In this paper, we first present an abstract model to conceptualize the essential functions of a market-based cluster RMS to support utility-driven cluster computing in practice. We then develop a taxonomy consisting of five sub-taxonomies, namely the *Market Model*, the *Resource Model*, the *Job Model*, the *Resource Allocation Model* and the *Evaluation Model* (see Figure 1). The taxonomy is applied in a survey to gain a better understanding of current research progress in developing effective market-based cluster RMSs.

The taxonomy not only helps to reveal key design factors and issues that are still outstanding and crucial but also provide insights for extending and reusing components of existing market-based RMSs. Therefore, the taxonomy can lead towards more practical and enhanced market-based cluster RMSs being designed and implemented in future.

The market-based RMSs selected for the survey are primarily research work as they reflect the latest technological advances. The design concepts and architectures of these research-based RMSs are also well-documented in publications to facilitate comprehensive comparisons, unlike commercially released products by vendors.

2. RELATED WORK

There are several proposed taxonomies for scheduling in distributed and heterogeneous computing. However, none of these taxonomies focus on market-based cluster computing environments. The taxonomy in [38] classifies scheduling strategies for general-purpose distributed systems. In [39], two taxonomies for state estimation and decision making are proposed to characterize dynamic scheduling for distributed systems. The EM^3 taxonomy in [40] utilizes the number of different execution modes and machine models to identify and classify heterogeneous systems. In [41], a modified version of the scheduling taxonomy in [40] is proposed to describe the resource allocation of heterogeneous systems. The taxonomy in [42] considers three characteristics of heterogeneous systems: application model; platform model; and mapping strategy to define resource matching and scheduling. A taxonomy on Grid RMS [43] includes a scheduling sub-taxonomy that examines four scheduling characteristics: scheduler organization; state estimation; rescheduling; and scheduling policy. However, our taxonomy focuses on market-based cluster RMSs for utility-driven cluster computing where cluster systems have a number of significant differences compared with Grid systems. One key difference is that a cluster system is distributed within a single administrative domain, whereas a Grid system is distributed across multiple administrative domains.

3. ABSTRACT MODEL FOR MARKET-BASED CLUSTER RESOURCE MANAGEMENT SYSTEM

Figure 2 outlines an abstract model for the market-based cluster RMS. The purpose of the abstract model is to identify generic components that are fundamental and essential in a practical market-based

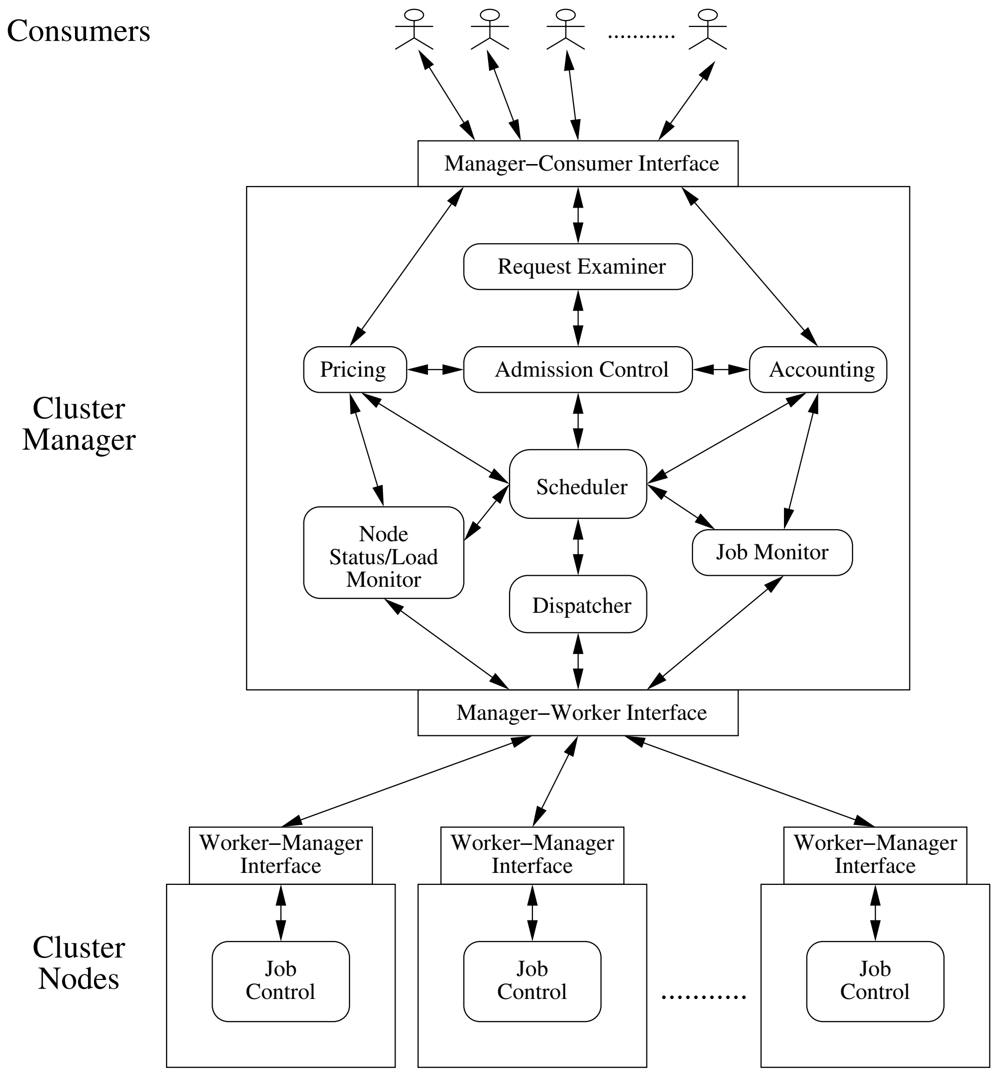

Figure 2. The abstract model for market-based cluster RMSs.

cluster RMS and portray the interactions between these components. Thus, the abstract model can be used to study how existing cluster RMS architectures can be leveraged and extended to incorporate market-based mechanisms to support utility-driven cluster computing in practice.

The market-based cluster RMS consists of two primary entities: a cluster manager; and a cluster node. For implementations within cluster systems, the machine that operates as the cluster manager can be known as the manager, server or master node and the machine that operates as the cluster node can be known as the worker or execution node. The actual number of cluster manager and cluster nodes depends on the implemented management control. For instance, a simple and common configuration for cluster systems is to support centralized management control where a single cluster manager collates multiple cluster nodes into a pool of resources as shown in Figure 2.

The cluster manager serves as the front-end for users and provides the scheduling engine responsible for allocating cluster resources to user applications. Thus, it supports two interfaces: the manager–consumer interface to accept requests from consumers; and the manager–worker interface to execute requests on selected cluster nodes. The consumers can be actual user applications, service brokers that act on the behalf of user applications or other cluster RMSs such as those operating in multi-clustering or Grid federation environments where requests that cannot be fulfilled locally are forwarded to other cooperative clusters.

When a service request is first submitted, the request examiner interprets the submitted request for QoS requirements such as deadline and budget. The admission control then determines whether to accept or reject the request in order to ensure that the cluster system is not overloaded whereby many requests cannot be fulfilled successfully. The scheduler selects suitable worker nodes to satisfy the request and the dispatcher starts the execution on the selected worker nodes. The node status/load monitor keeps track of the availability of the nodes and their workload, while the job monitor maintains the execution progress of requests.

It is vital for a market-based cluster RMS to support pricing and accounting mechanisms. The pricing mechanism decides how requests are charged. For instance, requests can be charged based on submission time (peak/off-peak), pricing rates (fixed/changing) or availability of resources (supply/demand). Pricing serves as a basis for managing the supply and demand of cluster resources and facilitates in prioritizing resource allocations effectively. The accounting mechanism maintains the actual usage of resources by requests so that the final cost can be computed and charged to the consumers. In addition, the maintained historical usage information can be utilized by the scheduler to improve resource allocation decisions.

The cluster nodes provide the resources for the cluster system to execute service requests via the worker–manager interface. The job control ensures that requests are fulfilled by monitoring execution progress and enforcing resource assignment for executing requests.

4. TAXONOMY

The taxonomy emphasizes on the practical aspects of market-based cluster RMSs that are vital to achieve utility-driven cluster computing in practice. It identifies key design factors and issues based on five major perspectives, namely the *Market Model*, the *Resource Model*, the *Job Model*, the *Resource Allocation Model* and the *Evaluation Model*.

4.1. Market Model taxonomy

The *Market Model* taxonomy examines how market concepts present in real-world human economies are incorporated into market-based cluster RMSs. This allows developers to understand what market-related attributes need to be considered, and in particular, to deliver utility. The Market Model taxonomy comprises four sub-taxonomies: the *economic model*, the *participant focus*, the *trading environment* and *QoS attributes* (see Figure 3).

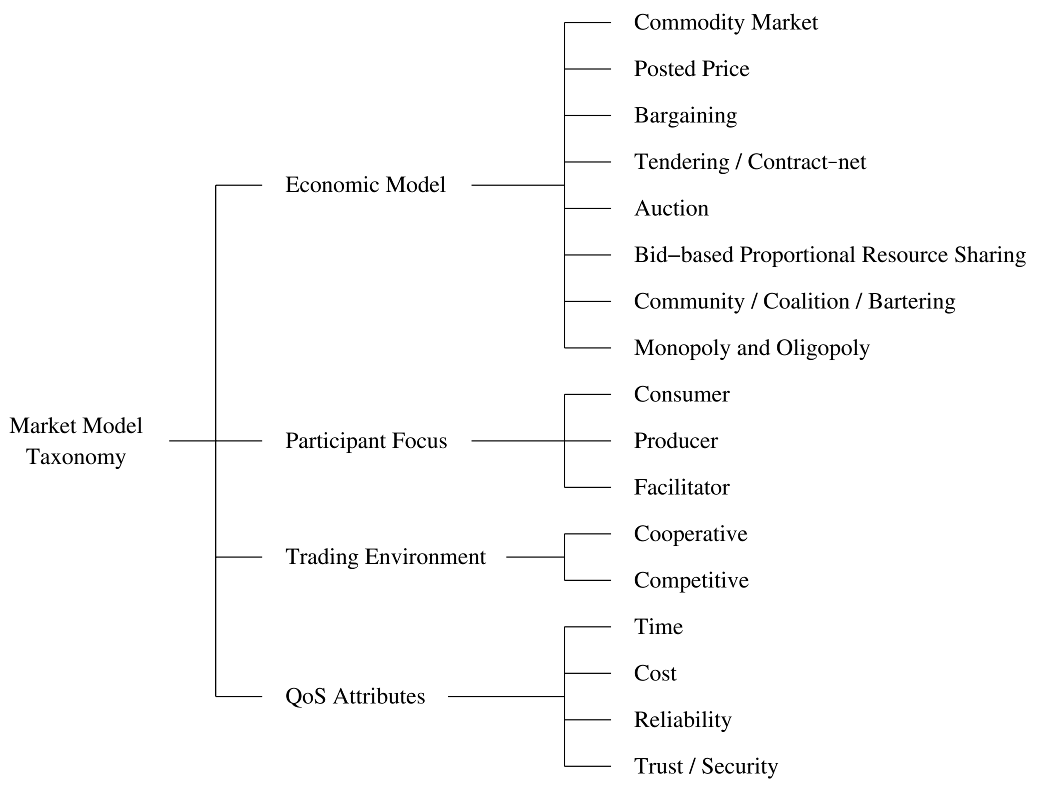

Figure 3. The Market Model taxonomy.

4.1.1. Economic model

The *economic model* derived from [44] establishes how resources are allocated in a market-driven computing environment. Selection of a suitable economic model primarily depends on the market interaction required between the consumers and producers.

In a *commodity market*, producers specify prices and consumers pay for the amount of resources they consume. The pricing of resources can be determined using various parameters, such as usage time and usage quantity. There can be flat or variant pricing rates. A flat rate means that pricing is fixed for a certain time period, whereas, a variant rate means that pricing changes over time, often based on the current supply and demand at that point of time. A higher demand results in a higher variant rate.

Posted price operates similarly to the commodity market. However, special offers are advertised openly so that consumers are aware of discounted prices and can thus utilize the offers. *Bargaining* enables both producers and consumers to negotiate for a mutually agreeable price. Producers typically start with higher prices to maximize profits but consumers start with lower prices to minimize costs.

Negotiation stops when the producer or consumer does not wish to negotiate further or a mutually agreeable price has been reached. Bargaining is often used when supply and demand prices cannot be easily defined.

In *tendering/contract-net*, the consumer first announces their requirements to invite bids from potential producers. Producers then evaluate the requirements and can respond with bids if they are interested and capable of the service or ignore the announcement if they are not interested or too busy. The consumer consolidates bids from potential producers, select the most suitable producer and sends a tender to the selected producer. The tender serves as a contract and specifies conditions that the producer has to accept and conform to. Penalties may be imposed on producers if the conditions are not met. The selected producer accepts the tender and delivers the required service. The consumer then notifies other producers of the unsuccessful outcome. Tendering/contract-net allows a consumer to locate the most suitable producer to meet its service request. However, it does not always guarantee locating the best producer each time since potential producers can choose not to respond or may be too busy.

Auction allows multiple consumers to negotiate with a single producer by submitting bids through an auctioneer. The auctioneer acts as the coordinator and sets the rules of the auction. Negotiation continues until a single clearing price is reached and accepted or rejected by the producer. Thus, auction regulates supply and demand based on the number of bidders, bidding price and offer price. There are basically five primary types of auctions, namely English, First-price, Vickrey, Dutch and Double [44].

Bid-based proportional resource sharing assigns resources proportionally, based on the bids given by the consumers. So, each consumer is allocated a proportion of the resources as compared with a typical auction model where only one consumer with the winning bid is entitled to the resource. This is ideal for managing a large shared resource where multiple consumers are equally entitled to the resource. *Community/coalition/bartering* supports a group of community producers/consumers who shares each others' resources to create a cooperative sharing environment. This model is typically adopted in computing environments where consumers are also producers and thus both contribute and use resources. Mechanisms are required to regulate that participants act fairly in both the roles of producers and consumers. *Monopoly/oligopoly* depicts a non-competitive market where only a single producer (monopoly) or a number of producers (oligopoly) determines the market price. Consumers are not able to negotiate or affect the stated price from the producers.

Most market-based cluster RMSs adopt an individual economic model directly. It is also possible to use hybrids or modified variants of the economic models in order to harness the strengths of different models and provide improved customizations based on user-specific application criteria. For example, the Stanford Peers Initiative [34] adopts a hybrid of auction and bartering economic models. Through an auction, the producer site selects the most beneficial consumer site with the lowest bid request for storage space, in exchange for its earlier request for storage space on the consumer site. This storage exchange between server and consumer sites creates a bartering system.

4.1.2. Participant focus

The *participant focus* identifies the party for whom the market-based cluster RMS aims to achieve benefit or utility. Having a *consumer* participant focus implies that a market-based cluster RMS aims to meet the requirements specified by cluster users and possibly optimize their perceived utility. For instance, the consumer may want to spend a minimal budget for a particular job.

Similarly, a *producer* participant focus results in resource owners fulfilling their desired utility. It is also possible to have a *facilitator* participant focus whereby the facilitator acts like an exchange and gains profit by coordinating and negotiating resource allocations between consumers and producers.

In utility-driven cluster computing, market-based cluster RMSs need to focus primarily on achieving utility for the consumers since their key purpose is to satisfy end-users' demand for service. For example, REXEC [23] maximizes utility for consumers by allocating processing time proportionally to jobs based on their bid values. However, producers and facilitators may have specific requirements that also need to be taken into consideration and not neglected totally. For instance, Cluster-On-Demand [25] has producer participant focus as each cluster manager maximizes its earnings by accessing the risk and reward of a new job before accepting it. It is also possible for market-based RMSs to have multiple participant focus. For example, the Stanford Peers Initiative [34] has both producer and consumer participant focuses as a site contributes storage space to other sites (producer) but also requests storage space in return from these sites (consumer).

4.1.3. Trading environment

The *trading environment* generalizes the motive of trading between the participants that are supported via the market-based cluster RMS. The needs and aims of various participants establish the trading relationships between them. In a *cooperative* trading environment, participants work together with one another to achieve a collective benefit for them, such as producers creating a resource sharing environment that speeds up the execution of jobs. On the other hand, in a *competitive* trading environment, each participant works towards its own individual benefit and does not take into account how they affect other participants, such as consumers contending with one another to secure available resources for their jobs.

A market-based cluster RMS can only support either a cooperative or competitive trading environment, but not both. For example, in Libra [24], consumers are awarded incentives to encourage submitting jobs with more relaxed deadlines so that jobs from other consumers can still be accepted. REXEC [23] creates a competitive trading environment whereby consumers are allocated proportions of processing time based on their bid values; a higher bid value entitles the consumer to a larger proportion.

4.1.4. QoS attributes

QoS attributes describe service requirements that consumers require the producer to provide in a service market. The *time* QoS attribute identifies the time required for various operations. Examples of time QoS attributes are job execution time, data transfer time and the deadline required by the consumer for the job to be completed. The *cost* QoS attribute depicts the cost involved for satisfying the job request of the consumer. A cost QoS attribute can be monetary, such as the budget that a consumer is willing to pay for the job to be completed or non-monetary, measured in units such as the data storage (in bytes) required for the job to be executed.

The *reliability* QoS attribute represents the level of service guarantee that is expected by the consumer. Jobs that require high reliability need the market-based cluster RMS to be highly fault-tolerant whereby check-pointing and backup facilities, with fast recovery after service failure are incorporated. The *trust/security* QoS attribute determines the level of security needed for executing

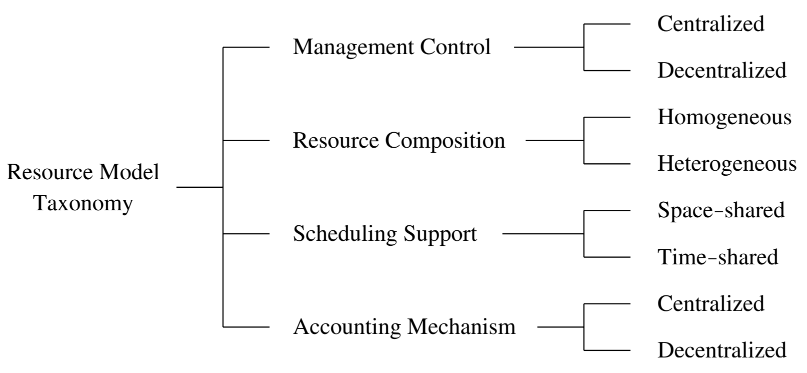

Figure 4. The Resource Model taxonomy.

applications on resources. Jobs with highly sensitive and confidential information require a resource with high trust/security to process.

Market-based cluster RMSs need to support time, cost and reliability QoS attributes as they are critical in enabling a service market for utility-driven cluster computing. The trust/security QoS attribute is also critical if the user applications require secure access to resources. For example, Libra [24] guarantees that jobs accepted into the cluster finish within the users' specified deadline (time QoS attribute) and budget (cost QoS attribute). There is no market-based cluster RMS that currently supports either reliability or trust/security QoS attributes.

Satisfying QoS attributes is highly critical in a service market as consumers pay based on the different levels of service required. The market-based cluster RMS should be able to manage service demands without sacrificing existing service requests and resulting in service failures. Failure to meet QoS attributes not only requires the producer to compensate consumers but also has a bad reputation on the producer that affects future credibility. For example, in Cluster-On-Demand [25] and LibraSLA [45], penalties are incurred for failing to satisfy the jobs' needs.

4.2. Resource Model taxonomy

The *Resource Model* taxonomy describes architectural characteristics of cluster systems. It is important to design market-based cluster RMSs that conform to the clusters' underlying system architectures and operating environments since there may be certain cluster system attributes that can be exploited. The Resource Model taxonomy comprises five sub-taxonomies: *management control*; *resource composition*; *scheduling support*; and *accounting mechanism* (see Figure 4).

4.2.1. Management control

The *management control* depicts how the resources are managed and controled in the cluster systems. A cluster with *centralized* management control has a single centralized resource manager that administers all the resources and jobs in the cluster. On the other hand, a cluster with *decentralized* management control has more than one decentralized resource manager managing subsets of resources within a cluster. Decentralized resource managers need to communicate with one another in order to be informed of local information of other managers.

A centralized resource manager collects and stores all local resource and job information within the cluster at a single location. Since a centralized resource manager has the global knowledge of the entire state of the cluster system, it is easier and faster for a market-based cluster RMS to communicate and coordinate with a centralized resource manager, as opposed to several decentralized resource managers. A centralized resource manager also allows a large change to be incorporated in the cluster environment as the change needs to be updated at a single location.

However, a centralized management control is more susceptible to bottlenecks and failures due to the overloading and malfunction of the single resource manager. A simple solution is to have backup resource managers that can be activated when the current centralized resource manager fails. In addition, centralized control architectures are less scalable compared to decentralized control architectures. Since centralized and decentralized management have various strengths and weaknesses, they perform better for different environments.

Centralized management control is mostly implemented in cluster systems since they are often owned by a single organization and modeled as a single unified resource. Therefore, it is sufficient for market-based cluster RMSs to assume centralized management control. For instance, Libra [24] extends upon the underlying cluster RMSs such as PBS [8] that implement centralized management control.

It may be appropriate for market-based cluster RMSs to have decentralized management control for better scalability and reliability. For example, in Enhanced MOSIX [46], each cluster node makes its independent resource allocation decisions, while in REXEC [23], multiple daemons separately discover and determine the best node for a job. Decentralized management control also facilitates federation of resources across multiple cluster systems. For example, in a cooperative and incentive-based coupling of distributed clusters [47], every cluster system has an inter-cluster coordinator in addition to the local cluster RMS to determine whether jobs should be processed locally or forwarded to other cluster systems.

4.2.2. *Resource composition*

The *resource composition* defines the combination of resources that make up the cluster system. A cluster system with a *homogeneous* resource composition consists of all worker nodes having the same resource components and configurations, whereas a *heterogeneous* resource composition consists of worker nodes having different resource components and configurations.

Most cluster systems have a homogeneous resource composition as it facilitates parallel processing of applications that require the same type of resources to process. In addition, processing is also simpler and much faster since there is no need to recompile the application for different resource configurations. Therefore by default, market-based cluster RMSs assumes a homogeneous resource composition.

However, it is possible that some cluster systems have a heterogeneous resource composition since heterogeneity can allow the concurrent execution of distinct applications that need different specific resources. These cluster systems may have different groups of worker nodes with homogeneous resource composition within each set for improved processing performance. Thus, it is ideal if market-based cluster RMSs can also support heterogeneous resource composition. These market-based cluster RMSs must also have an effective means of translating requirements and measurements such as required QoS attributes and load information across heterogeneous nodes to ensure accuracy. For example, in Enhanced MOSIX [46], usages of various resources in a node are translated into a single standard cost measure to support heterogeneity.

4.2.3. Scheduling support

The *scheduling support* determines the type of processing that is supported by the cluster's underlying operating system. The *space-shared* scheduling support enables only a single job to be executed at any one time on a processor, whereas the *time-shared* scheduling support allows multiple jobs to be executed at any one time on a processor. For example, most traditional cluster RMSs such as PBS [8] and SGE [9] support both space-shared and time-shared scheduling supports.

Depending on its resource allocation approach, a market-based cluster RMSs may require either a space-shared or a time-shared scheduling support. For a space-shared scheduling support, a started job can finish earlier since it has full individual access to the processor and is thus executed faster. However, submitted jobs also need to wait longer to be started for space-shared scheduling support if no processor is free, assuming that preemption is not supported.

On the other hand, time-shared scheduling support only allows shared access to processors but may reallocate unused processing time to other jobs if a job is not using the allocated processing power, such as when reading or writing data. Therefore, time-shared scheduling support can lead to a higher throughput of jobs over a period of time. Moreover, time-shared scheduling support is likely to incur less latency for executing multiple jobs when compared with a space-shared scheduling support that needs to preempt the current active job and start the new one. For instance, Libra [24], REXEC [23] and Tycoon [30] utilize time-shared scheduling support to share proportions of processing power among multiple active jobs.

4.2.4. Accounting mechanism

The *accounting mechanism* maintains and stores information about job executions in the cluster system. The stored accounting information may then be used for charging purposes or planning future resource allocation decisions. A *centralized* accounting mechanism denotes that information for the entire cluster system is maintained by a single centralized accounting manager and stored on a single node. For example, REXEC [23] has a centralized accounting service to maintain credit usage for each user.

A *decentralized* accounting mechanism indicates that multiple decentralized accounting managers monitor and store separate sets of information on multiple nodes. For instance, in Tycoon [30], each local host stores accounting information to compute service cost that users need to pay and determine prices of advance resource reservation for risk-averse jobs.

Similar to the management control taxonomy, it is easier for market-based cluster RMSs to access information based on the centralized accounting mechanism. However, the centralized accounting mechanism is less reliable and scalable compared with the decentralized accounting mechanism. For more flexibility and extensibility, market-based cluster RMSs can be designed to support both centralized and decentralized accounting mechanisms. Most current implementations of market-based

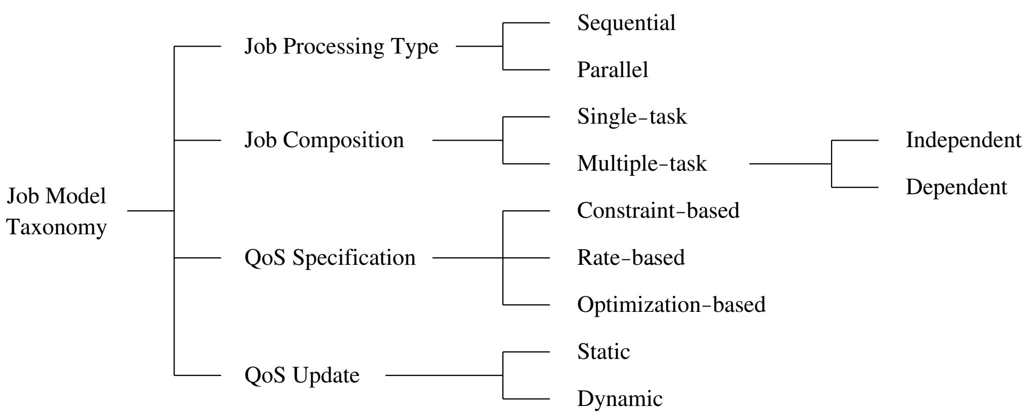

Figure 5. The Job Model taxonomy.

RMS already have built-in accounting mechanisms that maintain job execution information but may not support charging functionality which is critical to actually provide a service market.

A probable solution is to connect such market-based RMSs to specialized accounting mechanisms that support charging functionality, such as GridBank [48] and QBank [49]. In GridBank, each Grid resource uses a Grid resource meter to monitor the usage information and a GridBank charging module to compute the cost. The centralized GridBank server then transfers the payment from the users' bank accounts to the Grid resource's account. On the other hand, QBank supports both centralized and decentralized configurations. For instance, the simplest and most tightly-coupled centralized QBank configuration is having a central scheduler, bank server and database for all resources that are suitable for a cluster environment. QBank also allows multiple schedulers, bank servers and databases for each separate resource in different administrative domains to support a highly decentralized P2P or Grid environment.

4.3. Job Model taxonomy

The *Job Model* taxonomy categorizes attributes of jobs that are to be executed on the cluster systems. Market-based cluster RMSs need to take into account job attributes to ensure that different job types with distinct requirements can be fulfilled successfully. The Job Model taxonomy comprises five sub-taxonomies: *job processing type*; *job composition*; *QoS specification*; and *QoS update* (see Figure 5).

4.3.1. Job processing type

The *job processing type* describes the type of processing that is required by the job. For *sequential* job processing type, the job executes on one processor independently. For *parallel* job processing type, the parallel job has to be distributed to multiple processors before executing these multiple processes simultaneously. Thus, parallel job processing type speeds up processing and is often used for solving complex problems. One common type of parallel job processing type is called message-passing, where multiple processes of a parallel program on different processors interact with one another via sending and receiving messages.

All market-based cluster RMSs already support sequential job processing type, which is a basic requirement for job execution. However, they also need to support parallel job processing types since most computation-intensive jobs submitted to cluster systems require parallel job processor type to speed up processing of complex applications. For example, Enhanced MOSIX [46] and REXEC [23] supports parallel job processing type.

4.3.2. *Job composition*

The *job composition* portrays the collection of tasks within a job that is defined by the user. The *single-task* job composition refers to a job having a single task, while the *multiple-task* job composition refers to a job being composed of multiple tasks.

For multiple-task job composition, the tasks can be either *independent* or *dependent*. Independent tasks can be processed in parallel to minimize the overall processing time. For example, a parameter sweep job has independent multiple-task job composition since it is composed of multiple independent tasks, each with a different parameter.

On the other hand, tasks may depend on specific input data that are only available at remote locations and need to be transferred to the execution node or are not yet available as some other jobs waiting to be processed generate the data. This means that a dependent multiple-task job composition needs to be processed in a pre-defined manner in order to ensure that all its required dependencies are satisfied. For example, a workflow job has dependent multiple-task job composition. Directed Acyclic Graphs (DAG) are commonly used to visualize the required pre-defined order of processing for workflows with no cycles, whereas workflow specification languages such as Business Process Execution Language (BPEL) [50] and XML-based workflow language (xWFL) [14] can be used to define and process workflows.

It is essential for market-based cluster RMSs to support all three job compositions: single-task; independent multiple-task; and dependent multiple-task. Single-task job composition is a basic requirement and already supported by all market-based cluster RMSs. There are also many complex scientific applications and business processes that require independent multiple-task job composition, such as parameter-sweep or dependent multiple-task job composition, such as workflow for processing. For example, Nimrod/G [28] dispatches and executes parameter-sweep jobs on a Grid. Currently, there appears to be no market-based cluster RMS that can facilitate execution of parameter-sweep or workflow jobs on cluster systems. Therefore, supporting all these job compositions in a market-based cluster RMS expands the community of consumers that can utilize cluster resources.

More complex mechanisms are needed to support multiple-task job composition. The market-based cluster RMS needs to schedule and monitor each task within the job to ensure that the overall job requirements can still be met successfully. For dependent multiple-task job composition, it is important to prioritize different dependencies between tasks. For example, a parent task with more dependent child tasks needs to be processed earlier to avoid delays. It is also possible to execute independent sets of dependable tasks in parallel since tasks are only dependent on one another within a set and not across sets.

4.3.3. QoS specification

The *QoS specification* describes how users can specify their QoS requirements for a job to indicate their perceived level of utility. This provides cluster users with the capability to influence the resource allocation outcome in the cluster.

Users can define *constraint-based* QoS specifications that use a bounded value or a range of values for a particular QoS so that the minimal QoS requirements can be fulfilled. Some examples of constraint-based QoS specifications that users can specify are execution deadline, execution budget, memory storage size, disk storage size and processor power. For instance, a user can specify a deadline less than one hour (value) or deadline between one and two hours (range of values) for executing a job on cluster nodes with available memory storage size of more than 256 MB (value) and processor speed between 200 GHz and 400 GHz (range of values).

A *rate-based* QoS specification allows users to define constant or variable rates that signify the required level of service over time. For instance, a user can specify a constant cost depreciation rate of ten credits per minute such that the user pays less for a slower job completion time. To support a higher level of personalization, users can state *optimization-based* QoS specifications that identify a specific QoS to optimize in order to maximize the users' utility. For example, a user may optimize the deadline of their job so that the job can be completed in the shortest possible time.

Market-based cluster RMSs may need to provide any of the constraint-based, rate-based or optimization-based QoS specifications so that the required utility of consumers are considered and delivered successfully. For example, REXEC [23] allows a user to specify the maximum bid value that acts as a cost constraint for executing a job. Cluster-On-Demand [25] uses rate-based QoS specification where each job has a value function that depreciates at a constant rate to represent its urgency. Optimization-based QoS specification in Nimrod/G [28] minimizes time within deadline constraint or cost within deadline constraint.

SLAs need to be negotiated and fixed between a cluster system and its users to ensure that an expected level of service performance is guaranteed for submitted jobs. There are specially-designed service specification languages, such as Web Services Agreement Specification (WS-Agreement) [51], Web Service Level Agreement (WSLA) [52], and BPEL [50] that can be utilized by market-based cluster RMSs to interpret and enforce negotiated SLAs.

4.3.4. QoS update

The *QoS update* determines whether QoS requirements of jobs can change after jobs are submitted and accepted. The *static* QoS update means that the QoS requirements given during job submission remain fixed and do not change after the job is submitted, while the *dynamic* QoS update means that QoS requirements of the jobs can change. These dynamic changes may already be pre-defined during job submission or modified by the user during an interactive job submission session.

Currently, all market-based cluster RMSs assumes static QoS update. For example, Libra [24] and REXEC [23] only allow users to specify QoS constraints during initial job submission. Market-based cluster RMSs also need to support dynamic QoS updates so that users have the flexibility to update their latest QoS needs since it is possible that users have changing QoS needs over time. The market-based cluster RMS should also be able to reassess newly changed QoS requirements and revise resource assignments accordingly as previous resource assignments may be ineffective to meet the

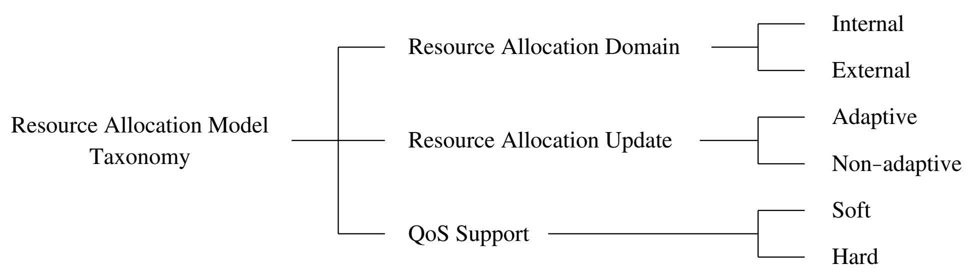
Resource Allocation Model Taxonomy
Resource Allocation Domain
Internal
External
Resource Allocation Update
Adaptive
Non-adaptive
QoS Support
Soft
Hard

Figure 6. The Resource Allocation Model taxonomy.

new requirements. In addition, it is highly probable that other planned or executing jobs may also be affected so there is a need to reassess and reallocate resources to minimize any possible adverse effects.

4.4. Resource Allocation Model taxonomy

The *Resource Allocation Model* taxonomy analyzes factors that can influence how the market-based cluster RMS operates and thus affect the resource assignment outcome. The Resource Allocation Model taxonomy comprises three sub-taxonomies: *resource allocation domain*, *resource allocation update* and *QoS support* (see Figure 6).

4.4.1. Resource allocation domain

The *resource allocation domain* defines the scope that the market-based cluster RMS is able to operate in. Having an *internal* resource allocation domain restricts the assignment of jobs to within the cluster system. An *external* resource allocation domain allows the market-based cluster RMS to assign jobs externally outside the cluster system, meaning that jobs can be executed on other remote cluster systems. Remote cluster systems may be in the same administrative domain belonging to the same producer such as an organization that owns several cluster systems or in different administrative domains owned by other producers such as several organizations that individually own some cluster systems. For instance, Cluster-On-Demand [25], Nimrod/G [28] and the Stanford Peers Initiative [34] allocate jobs externally to multiple remote systems, instead of internally.

Most market-based cluster RMSs often only support internal resource allocation domains. Supporting external resource allocation domains can otherwise allow a market-based cluster RMS to have access to more alternative resources to possibly satisfy more service requests. This results in higher flexibility and scalability as service requests can still be fulfilled when there are insufficient internal resources within the cluster systems to meet demands. Users thus benefit since their service requests are more likely to be fulfilled. Cluster owners can also earn extra revenues by accepting external service requests. Therefore, it is ideal if market-based cluster RMSs can support external resource allocation, in addition to internal resource allocation. However, there is also the need to address other issues for supporting external resource allocation domains such as data transfer times, network delays and reliability of remote cluster systems.

4.4.2. *Resource allocation update*

The *resource allocation* update identifies whether the market-based cluster RMS is able to detect and adapt to changes to maintain effective scheduling. *Adaptive* resource allocation updates means that the market-based cluster RMS is able to adjust dynamically and automatically to suit any new changes. For example, Libra [24] can allocate resources adaptively based on the actual execution and required deadline of each active job so that later arriving urgent jobs are allocated more resources.

On the other hand, *non-adaptive* resource allocation updates means that the RMS is not able to adapt to changes and thus still continue with its original resource assignment decision. For instance, in the Stanford Peers Initiative [34], the amount of storage space allocated for data exchange remains fixed and does not change once a remote site has been selected.

In actual cluster environments, the operating condition varies over time, depending on factors such as availability of resources, amount of submission workload and users' service requirements. An initially good resource assignment decision may lead to an unfavorable outcome if it is not updated when the operating scenario changes. Likewise, there is also a possibility of improving a previously poor resource allocation decision. Therefore, market-based cluster RMSs need to support adaptive resource allocation updates so that they are able to adjust and operate in changing situations to deliver a positive outcome. However, supporting adaptive resource allocation updates also requires effective mechanisms to detect and determine when and how to adapt to various scenarios.

4.4.3. *QoS support*

The *QoS support* derived from [43] determines whether the QoS specified by the user can be achieved. The *soft* QoS support allows user to specify QoS parameters but do not guarantee that these service requests can be satisfied. For example, Nimrod/G [28] provides soft QoS support as it adopts a best effort approach to schedule jobs and stop scheduling once their QoS constraints are violated.

On the contrary, the *hard* QoS support is able to ensure that the specified service can definitely be achieved. Examples of market-based cluster RMSs that provide hard QoS support are Libra [24] and REXEC [23]. Libra guarantees that accepted jobs are finished within their deadline QoS, while REXEC ensures that job execution costs are limited to users' specified cost QoS.

The QoS support that a market-based cluster RMS provides should correspond to the users' service requirements. An example is the deadline QoS parameter. If a user requires a hard deadline for an urgent job, it is pointless for a market-based cluster RMSs employing soft QoS support to accept the job as it does not guarantee that the deadline can be met. In reality, since different users often have various service requirements, it is best to have a market-based cluster RMSs that can provide both soft and hard QoS supports. The market-based cluster RMS can then satisfy more service requests as soft service requests can be accommodated without compromising hard QoS requests. Jobs with a soft deadline may be delayed so that more jobs with hard deadline can be satisfied.

Admission control is essential during job submission to determine and feedback to the user whether the requested hard or soft QoS can be given. If accepted by the admission control, jobs requiring hard

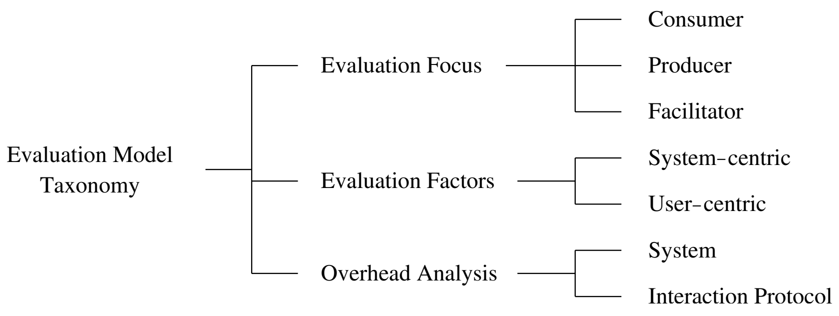

Figure 7. The Evaluation Model taxonomy.

QoS support need to be monitored to ensure that the required QoS is enforced and fulfilled. This is non-trivial as a high degree of coordination and monitoring may be necessary to enforce the QoS.

With the incorporation of penalties in SLAs, it becomes increasingly important for market-based cluster RMSs to deliver the required QoS as requested. Failure to deliver the agreed level of service can thus result in penalties that lower the financial benefits of the cluster owners. For example, penalties are modeled in Cluster-On-Demand [25] and LibraSLA [45]. In Cluster-On-Demand, jobs are penalized if they finish later than their required runtimes, whereas in LibraSLA, jobs are penalized after the lapse of their deadlines.

4.5. Evaluation Model taxonomy

The *Evaluation Model* taxonomy outlines how to assess the effectiveness and efficiency of market-based RMSs for utility-driven cluster computing. The Evaluation Model taxonomy comprises three sub-taxonomies: *evaluation focus*, *evaluation factors* and *overhead analysis* (see Figure 7).

4.5.1. Evaluation focus

The *evaluation focus* identifies the party that the market-based cluster RMS is supposed to achieve utility for. The *consumer* evaluation focus measures the level of utility that has been delivered to the consumer based on its requirements. Likewise, the *producer* and *facilitator* evaluation focus evaluates how much value is gained by the producer and facilitator respectively. For example, Libra [53] evaluates the utility achieved for consumers (users) via the Job QoS Satisfaction metric and the benefits gained by the producer (cluster owner) via the Cluster Profitability metric.

The evaluation focus is similar to the participant focus sub-taxonomy discussed previously since it is logical to measure performance based on the selected participant focus. It is important to verify whether the market-based cluster RMS is able to achieve utility for the selected participant focus as expected. The evaluation focus also facilitates comparisons between market-based cluster RMSs with the same participant focus, enabling them to be identified and evaluated relative to one another to establish similarities and differences.

4.5.2. Evaluation factors

Evaluation factors are metrics defined to determine the effectiveness of different market-based cluster RMSs. *System-centric* evaluation factors measure performance from the system perspective and thus depict the overall operational performance of the cluster system. Examples of system-centric evaluation factors are average waiting time, average response time, system throughput and resource utilization. Average waiting time is the average time that a job has to wait before commencing execution, while average response time is the average time taken for a job to be completed. System throughput determines the amount of work completed in a period of time, whereas resource utilization reflects the usage level of the cluster system. For instance, Libra [53] provides the average waiting time and the average response time as its system-centric evaluation factors.

User-centric evaluation factors assess performance from the participant perspective and thus portray the utility achieved by the participants. Different user-centric evaluation factors can be defined for assessing different participants that include consumer, producer or facilitator (as defined in the evaluation focus sub-taxonomy). For instance, Libra [53] defines the Job QoS Satisfaction evaluation factor for consumer evaluation focus and the Cluster Profitability evaluation factor for producer evaluation focus respectively. It is apparent that user-centric evaluation factors should constitute QoS attributes (as described in the QoS attributes sub-taxonomy) in order to assess whether the QoS required by consumers is attained. For example in Libra [53], the Job QoS Satisfaction evaluation factor computes the proportion of submitted jobs where the deadline and budget QoS parameters (time and cost in QoS attributes sub-taxonomy) are fulfilled, whereas the Cluster Profitability evaluation factor calculates the proportion of profit earned out of the total budget (cost in QoS attributes sub-taxonomy) of submitted jobs.

Both system-centric and user-centric evaluation factors are required to accurately determine the effectiveness of the market-based cluster RMS. System-centric evaluation factors ensure that system performance is not compromised entirely due to the emphasis on achieving utility, whereas user-centric evaluation factors prove that the market-based cluster RMS is able to achieve the required utility for various participants. Evaluation factors can also serve as benchmarks to grade how various market-based cluster RMSs perform for specific measures and rank them accordingly.

4.5.3. Overhead analysis

The *overhead analysis* examines potential overheads that are incurred by the market-based cluster RMS. The *system* overhead analysis considers overheads sustained by the market-based cluster RMS that are of system nature. Examples of system overhead are hard disk space, memory size and processor runtime. There seems to be no market-based RMSs that explicitly mention about system overhead analysis since it is often considered to be an intrinsic system implementation issue.

The *interaction protocol* overhead analysis determines overheads that are generated by the operating policies of the market-based cluster RMS. Examples of interaction protocol overhead are communications with various nodes to determine whether they are busy or available and derive the appropriate schedule of jobs to execute on them. For instance, Tycoon [30] addresses interaction protocol overhead analysis by holding auctions internally within each service host to reduce communication across hosts.

Overheads result in system slowdowns and can create bottlenecks, thus leading to poor efficiency. There is thus a need to analyze both system and interaction protocol overheads incurred by the market-based cluster RMS in order to ensure that any overheads are kept to the minimum or within manageable limits. Otherwise, a high system overhead adds unnecessary load that burdens the cluster system and reduces overall available resources to handle actual job processing. Whereas, a high interaction protocol overhead can result in longer communication times and unnecessary high network traffic that can slow down data transfers for executions. Lowering both system and interaction protocol overheads thus leads to a higher scalability for handling larger numbers of requests, which is critical for constructing viable market-based cluster RMSs.

5. SURVEY

Table I shows a summary listing of existing market-based RMSs that has been proposed by researchers for various computing platforms. Supported computing platforms include clusters, distributed databases, Grids, parallel and distributed systems, peer-to-peer and World Wide Web. In this section, we utilize the taxonomy to survey some of these existing market-based RMSs (denoted by * in Table I).

The market-based RMSs for the survey are chosen based on several criteria. First, the survey should be concise and include sufficient number of market-based RMSs to demonstrate how the taxonomy can be applied effectively. Second, the selected market-based RMSs are fairly recent works so that the survey creates an insight into the latest research developments. Finally, the selected market-based RMSs are relevant for this paper.

Market-based RMSs chosen for the survey can be classified into two broad categories: those proposed for cluster platforms; and those proposed for other computing platforms. Since the context of this paper focuses on utility-driven cluster computing, four market-based cluster RMSs (Cluster-On-Demand [25], Enhanced MOSIX [46], Libra [24] and REXEC [23]) are surveyed to understand current technological advances and identify outstanding issues that are yet to be explored so that more practical market-based cluster RMSs can be implemented in future.

On the other hand, surveying market-based RMSs for other computing platforms allows us to analyze and examine the applicability and suitability of these market-based RMSs for supporting utility-driven cluster computing in practice. This in turn helps us to identify possible strengths of these market-based RMSs that may be leveraged for cluster computing environments. We have chosen three market-based RMSs (Faucets [29], Nimrod/G [28] and Tycoon [30]) from Grids and one market-based RMS (the Stanford Peers Initiative [34]) from peer-to-peer since both Grids and peer-to-peer computing platforms are the latest and most active research areas that encompasses cluster systems distributed at multiple remote sites.

The survey using the various sub-taxonomies is summarized in the following tables: the Market Model (Table II); the Resource Model (Table III); the Job Model (Table IV); the Resource Allocation Model (Table V); and the Evaluation Model (Table VI). The 'NA' keyword in the tables denotes that either the specified sub-taxonomy is not addressed by the particular RMS or there is not enough information from the references to determine otherwise.

Table I. Summary of market-based resource management systems.

Computing platform	Market-based RMS	Economic model	Brief description
Clusters	Cluster-On-Demand * [25]	Tendering/ contract-net	Each cluster manager uses a heuristic to measure and balance the future risk of profit lost for accepting a job later against profit gained for accepting the job now
	Enhanced MOSIX * [46]	Commodity market	Uses process migration to minimize the overall execution cost of machines in the cluster
	Libra * [24]	Commodity market	Provides incentives to encourage users to submit job requests with longer deadlines
	REXEC * [23]	Bid-based proportional resource sharing	Allocates resources proportionally to competing jobs based on their users' valuation
	Utility Data Center [54]	Auction	Compares two extreme auction-based resource allocation mechanisms: a globally optimal assignment market mechanism; and a sub-optimal simple market mechanism
Distributed Databases	Anastasiadi *et al*. [26]	Posted price	Examines the scenario of load balancing economy where servers advertise prices at a bulletin board and transaction requests are routed based on three different routing algorithms that focus on expected completion time and required network bandwidth
	Mariposa [27]	Tendering/ contract-net	Completes a query within its user-defined budget by contracting portions of the query to various processing sites for execution
Grids	Bellagio [55]	Auction	A centralized auctioneer computes bid values based on number of requested resources and their required durations, before clearing the auctions at fixed time periods by allocating to higher bid values first
	CATNET [56]	Bargaining	Each client uses a subjective market price (computing using price quotes consolidated from available servers) to negotiate until a server quotes an acceptable price
	Faucets * [29]	Tendering/ contract-net	Users specify QoS contracts for adaptive parallel jobs and Grid resources compete for jobs via bidding

Table I. *Continued.*

Computing platform	Market-based RMS	Economic model	Brief description
Grids	G-commerce [57]	Commodity market, auction	Compares resource allocation using either commodity market or auction strategy based on four criteria: price stability; market equilibrium; consumer efficiency; and producer efficiency
	Gridbus [22]	Commodity market	Considers the data access and transfer costs for data-oriented applications when allocating resources based on time or cost optimization
	Gridmarket [58]	Auction	Examines resource allocation using Double auction where consumers set ceiling prices and sellers set floor prices
	Grosu and Das [59]	Auction	Studies resource allocation using First-price, Vickrey and Double auctions
	Maheswaran *et al*. [60]	Auction	Investigates resource allocation based on two 'co-bid' approaches that aggregate similar resources: first or no preference approaches
	Nimrod/G * [28]	Commodity market	Allocates resources to task farming applications using either time or cost optimization with deadline- and budget-constrained algorithms
	OCEAN [61]	Bargaining, tendering/ contract-net	First discovers potential sellers by announcing a buyer's trade proposal and then allows the buyer to determine the best seller by using two possible negotiation mechanisms: yes/no and static bargain
	Tycoon * [30]	Auction	Allocates resources using 'auction share' that estimates proportional share with consideration for latency-sensitive and risk-averse applications
Parallel and Distributed Systems	Agoric Systems [62]	Auction	Employs the 'escalator' algorithm where users submit bids that escalate over time based on a rate and the server uses a Vickrey auction at fixed intervals to award resources to the highest bidder who is then charged with the second-highest bid
	D'Agents [33]	Bid-based proportional resource sharing	The server assigns resources by computing the clearing price based on the aggregate demand function of all its incoming agents

Table I. *Continued.*

Computing platform	Market-based RMS	Economic model	Brief description
Parallel and Distributed Systems	Dynasty [63]	Commodity market	Uses a hierarchical-based brokering system where each request is distributed up the hierarchy until the accumulated brokerage cost is limited by the budget of the user
	Enterprise [31]	Tendering/ contract-net	Clients broadcast a request for bids with task description and select the best bid, which is the shortest estimated completion time given by available servers
	Ferguson *et al*. [64]	Posted price, auction	Examines how First-price and Dutch auctions can support a load balancing economy where each server host its independent auction and users decide which auction to participate based on last clearing prices advertised in bulletin boards
	Kurose and Simha [65]	Bid-based proportional resource sharing	Uses a resource-directed approach where the current allocation of a resource is readjusted proportionally according to the marginal values computed by every agent using that resource to reflect the outstanding quantity of resource needed
	MarketNet [66]	Posted price	Advertises resource request and offer prices on a bulletin board and uses currency flow to restrict resource usage so that potential intrusion attacks into the information systems are controled and damages caused are kept to the minimum
	Preist *et al*. [67]	Auction	An agent participates in multiple auctions selling the same goods in order to secure the lowest bid possible to acquire a suitable number of goods for a buyer
	Spawn [32]	Auction	Sub-divides each tree-based concurrent program into nodes (sub-programs) that then hold Vickrey auction independently to obtain resources
	Stoica *et al*. [68]	Auction	The job with the highest bid starts execution instantly if the required number of resources are available; else it is scheduled to wait for more resources to be available and has to pay for holding on to currently available resources

Table I. *Continued.*

Computing platform	Market-based RMS	Economic model	Brief description
Parallel and Distributed Systems	WALRAS [69]	Auction	Consumer and producer agents submit their demand and supply curves respectively for goods and the equilibrium price is determined through an iterative auctioning process
Peer-to-Peer	Stanford Peers Initiative * [34]	Auction, bartering	Uses data trading to create a replication network of digital archives where a winning remote site offers the lowest bid for free space on the local site in exchange for the amount of free space requested by the local site on the remote site
World Wide Web	Java Market [70]	Commodity market	Uses a cost-benefit framework to host an internet-wide computational market where producers (machines) are paid for executing consumers' jobs (Java programs) as Java applets in their Web browsers
	JaWS [37]	Auction	Uses a Double auction to award a lease contract between a client and a host that contains the following information: agreed price; lease duration; compensation; performance statistics vector; and abort ratio
	POPCORN [36]	Auction	Each buyer (parallel programs written using POPCORN paradigm) submits a price bid and the winner is determined through one of three implemented auction mechanisms: Vickrey, Double and Clearinghouse Double auctions
	SuperWeb [35]	Commodity market	Potential hosts register with client brokers and receive payments for executing Java codes depending on the QoS provided
	Xenoservers [71]	Commodity market	Supports accounted execution of untrusted programs such as Java over the Web where resources utilized by the programs are accounted and charged to the users

Table II. Survey using the Market Model taxonomy.

Market-based RMS	Economic model	Participant focus	Trading environment	QoS attributes
Cluster-On-Demand	Tendering/ contract-net	Producer	Competitive	Cost
Enhanced MOSIX	Commodity market	Producer	Cooperative	Cost
Libra	Commodity market	Consumer	Cooperative	Time, cost
REXEC	Bid-based proportional resource sharing	Consumer	Competitive	Cost
Faucets	Tendering/ contract-net	Producer	Competitive	Time, cost
Nimrod/G	Commodity market	Consumer	Competitive	Time, cost
Tycoon	Auction	Consumer	Competitive	Time, cost
Stanford Peers Initiative	Auction, bartering	Consumer, producer	Cooperative	Cost

Table III. Survey using the Resource Model taxonomy.

Market-based RMS	Management control	Resource composition	Scheduling support	Accounting mechanism
Cluster-On-Demand	Decentralized	NA	NA	Decentralized
Enhanced MOSIX	Decentralized	Heterogeneous	Time-shared	Decentralized
Libra	Centralized	Heterogeneous	Time-shared	Centralized
REXEC	Decentralized	NA	Time-shared	Centralized
Faucets	Centralized	NA	Time-shared	Centralized
Nimrod/G	Decentralized	Heterogeneous	NA	Decentralized
Tycoon	Decentralized	Heterogeneous	Time-shared	Decentralized
Stanford Peers Initiative	Decentralized	NA	NA	NA

Table IV. Survey using the Job Model taxonomy.

Market-based RMS	Job processing type	Job composition	QoS specification	QoS update
Cluster-On-Demand	Sequential	Independent single-task	Rate-based	Static
Enhanced MOSIX	Parallel	NA	NA	NA
Libra	Sequential	Independent single-task	Constraint-based	Static
REXEC	Parallel, sequential	Independent single-task	Constraint-based	Static
Faucets	Parallel	NA	Constraint-based	Static
Nimrod/G	Sequential	Independent multiple-task	Optimization-based	Static
Tycoon	NA	NA	Constraint-based	Static
Stanford Peers Initiative	NA	NA	NA	NA

Table V. Survey using the Resource Allocation Model taxonomy.

Market-based RMS	Resource allocation domain	Resource allocation update	QoS support
Cluster-On-Demand	External	Adaptive	Soft
Enhanced MOSIX	Internal	Adaptive	NA
Libra	Internal	Adaptive	Hard
REXEC	Internal	Adaptive	Hard
Faucets	Internal	Adaptive	Soft
Nimrod/G	External	Adaptive	Soft
Tycoon	Internal	Adaptive	Soft
Stanford Peers Initiative	External	Non-adaptive	NA

Table VI. Survey using the Evaluation Model taxonomy.

Market-based RMS	Evaluation focus	Evaluation factors	Overhead analysis
Cluster-On-Demand	Producer	User-centric (cost)	NA
Enhanced MOSIX	Consumer	User-centric (time)	NA
Libra	Consumer, producer	System-centric, user-centric (time, cost)	NA
REXEC	Consumer	User-centric (cost)	NA
Faucets	NA	NA	NA
Nimrod/G	NA	NA	NA
Tycoon	Consumer	User-centric (time)	Interaction protocol
Stanford Peers Initiative	Consumer, producer	User-centric (reliability)	NA

5.1. Cluster-On-Demand

Cluster-On-Demand (COD) [72] allows the cluster manager to dynamically create independent partitions called virtual clusters (vclusters) with specific software environments for each different user groups within a cluster system. This in turn facilitates external policy managers and resource brokers in the Grid to control their assigned vcluster of resources. A later work [25] examines the importance of opportunity cost in a service market where the earnings for a job depreciate linearly over an increasing time delay. A falling earning can become zero and instead become a penalty for not fulfilling the contract of task execution. Thus, each local cluster manager needs to determine the best job mix to balance the gains and losses for selecting one task ahead of another.

The task assignment among various cluster managers adopts the tendering/contract-net economic model. A user initiates an announcement bid that reflects its valuation for the task to all the cluster managers. Each cluster manager then considers the opportunity cost (gain or loss) for accepting the task and proposes a contract with an expected completion time and price. The user then selects and accepts a contract from the cluster manager that responded.

A competitive trading environment with producer participant focus is supported since each cluster manager aims to maximize their earnings by assessing the risk and reward for bidding and scheduling a task. Earnings are paid by users to cluster managers as costs for adhering to the conditions of the contract. All cluster managers maintain information about their committed workload in order to evaluate whether to accept or reject a new task, hence exercising decentralized management control and accounting mechanism.

Tasks to be executed are assumed to single and sequential. For each task, the user provides a value function containing a constant depreciation rate to signify the importance of the task and thus the required level of service. The value function remains static after the contract has been accepted by the user. Tasks are scheduled externally to cluster managers in different administrative domains. Adaptive resource allocation updates are supported as the cluster manager may delay less costly committed tasks for more costly new tasks that arrive later to minimize its losses for penalties incurred. This means that soft QoS support is provided since accepted tasks may complete later than expected.

Performance evaluation focuses on a producer by using a user-centric cost evaluation factor to determine the average yield or earning each cluster manager achieves. Simulation results show that considering and balancing the potential gain of accepting a task instantly with the risk of future loss provides better returns for competing cluster managers.

5.2. Enhanced MOSIX

Enhanced MOSIX [46] is a modified version of MOSIX [73] cluster operating system that employs an opportunity cost approach for load balancing to minimize the overall execution cost of the cluster. The opportunity cost approach computes a single marginal cost of assigning a process to a cluster node based on the processor and memory usages of the process, thus representing a commodity market economic model. The cluster node with the minimal marginal cost is then assigned the process. This implies a cooperative trading environment with producer participant focus whereby the cost utility is measured in terms of usage level of resources.

In Enhanced MOSIX, decentralized resource control is established where each cluster node makes its independent resource assignment decisions. Heterogeneous resource composition is supported by translating usages of different resources into a single cost measure.

Enhanced MOSIX supports a time-sharing parallel execution environment where a user can execute a parallel application by first starting multiple processes on one cluster node. Each cluster node maintains accounting information about processes on its node and exchanges information with other nodes periodically to determine which processes can be migrated, based on the opportunity cost approach. Process migration is utilized internally within the cluster to assign or reassign processes to less loaded nodes, hence supporting adaptive resource allocation updates.

Enhanced MOSIX does not address how QoS can be supported for users. For performance evaluation, it measures the slowdown of user processes, hence using a user-centric time evaluation factor. Simulation results show that using the opportunity cost approach returns a lower average slowdown of processes, thus benefiting the consumers.

5.3. Libra

Libra [24] is designed to be a pluggable market-based scheduler that can be integrated into existing cluster RMS architectures to support the allocation of resources based on users' QoS requirements. Libra adopts the commodity market economic model that charges users using a pricing function. A later work [53] proposes an enhanced pricing function that supports four essential requirements for pricing of utility-driven cluster resources: flexibility; fairness; dynamism; and adaptivity.

The pricing function is flexible to allow easy configuration of the cluster owner to determine the level of sharing. It is also fair as resources are priced based on actual usage; jobs that use more resources are charged more. The price of resources is dynamic and is not based on a static rate. In addition, the price of resources adapts to the changing supply and demand of resources. For instance, a high cluster workload results in an increase in pricing to discourage users from submitting infinitely and thus not overloading the cluster. This is crucial in providing QoS support since an overloaded cluster is not able to fulfill QoS requirements. In addition, incentive is given to promote users to submit jobs with longer deadlines; a job with longer deadline is charged less compared to a job with shorter deadline.

The main objective of Libra is to maximize the number of jobs whose QoS requirements can be met, thus enabling a consumer participant focus. The enhanced pricing function [53] also improves utility for the producer (cluster owner) as only jobs with higher budgets are accepted with increasing cluster workload. Libra also considers both time and cost QoS attributes by allocating resources based on the deadline and budget QoS parameters for each job. A cooperative trading environment is implied as users are encouraged to provide a more relaxed deadline through incentives so that more jobs can be accommodated.

Libra communicates with the centralized resource manager in the underlying cluster RMS that collects information about resources in the cluster system. For a heterogeneous resource composition, measures such as estimated execution time are translated to their equivalent on different worker nodes. The cluster RMS needs to support time-shared execution given that Libra allocates resources to multiple executing jobs based on their required deadline. This ensures that a more urgent job closer to its deadline is allocated a larger processor time partition on a worker node when compared with a less urgent job. Libra uses a centralized accounting mechanism to monitor resource usage of active jobs so as to periodically reallocate the time partitions for each active job to ensure all jobs still complete within their required deadline.

Libra currently assumes that submitted jobs are sequential and single-task. Users can express two QoS constraints: the deadline that the job needs to be completed; and the budget that the user is willing to pay. The QoS constraints cannot be updated after the job has been accepted for execution. Libra only schedules jobs to internal worker nodes within the cluster system. Each worker node has a job control component that reassigns processor time partitions periodically, based on the actual execution and required deadline of each active job and thus enforcing hard QoS support. This means that Libra can allocate resources adaptively to meet the deadline of later arriving but more urgent jobs.

Libra uses average waiting time and average response time as system-centric evaluation factors to evaluate overall system performance. In addition, Libra defines two user-centric evaluation factors [53]: Job QoS Satisfaction; and Cluster Profitability to measure the level of utility achieved for the consumers (users) and producer (cluster owner) respectively. The Job QoS Satisfaction determines the percentage of jobs where the deadline and budget QoS requirements are satisfied and thus examines the time and cost utility of the consumers. On the other hand, the Cluster Profitability calculates the proportion of profit obtained by the cluster owner and thus studies the cost utility of the producer. Simulation results show that Libra performs better than the traditional First-Come-First-Served scheduling approach for both system-centric and user-centric evaluation factors.

5.4. REXEC

REXEC [23] implements bid-based proportional resource sharing where users compete for shared resources in a cluster. REXEC has a consumer participant focus since resources are allocated proportionally, based on costs that competing users are willing to pay for a resource. Costs are defined as rates, such as credits per minute to reflect the maximum amount that a user wants to pay for using the resource.

Decentralized management control is achieved by having multiple daemons to separately discover and determine the best node to execute a job and then allowing each REXEC client to directly manage the execution of its jobs on the selected cluster nodes. The cluster nodes support time-shared scheduling support so that multiple jobs share resources at the same time. A centralized accounting service maintains credit usage for each user in the cluster. REXEC does not consider the resource composition since it determines the proportion of resource assignment for a job purely on its user's valuation.

REXEC supports the execution of both sequential and parallel programs. Users specify constraint-based cost limits that they are willing to spend and remains static after a job submission. The discovery and selection of nodes internal to the cluster system is designed to be independent so that users have the flexibility to determine the node selection policy through their own REXEC client. Existing resource assignments are recomputed whenever a new job starts or finishes on a node, thus enabling adaptive resource allocation updates. REXEC only considers a single QoS requirement where the cost of a job execution is limited to the users' specified rate. For a parallel program, the total credit required by all its processes is enforced not to exceed the cost specified by the user.

A later work [74] uses a user-centric evaluation factor; an aggregate utility that adds up all the users' costs for completing jobs on the cluster. The cost charged to the user depends on the completion time of his job and decreases linearly over time until it reaches zero. Therefore, this presents a consumer evaluation focus where cost is the evaluation factor.

5.5. Faucets

Faucets [29] aims to provide efficient resource allocation on the computational Grid for parallel jobs by improving its usability and utilization. For better usability, users do not need to manually discover the best resources to execute their jobs or monitor the progress of executing jobs. To improve utilization, the parallel jobs are made adaptive using Charm++ [75] or adaptive MPI [76] frameworks so that they can be executed on a changing number of allocated processors during runtime on demand [77]. This allows more jobs to be executed at any one time and no processors are left unused.

Market economy is implemented to promote utilization of the computational Grid where each individual Grid resource maximizes its profit through maximum resource utilization. For each parallel job submitted, the user has to specify its QoS contract that includes requirements such as the software environment, number of processors (can be a single number, a set of numbers or range of numbers), expected completion time (and how this changes with number of processors) and the payoff that the user pays the Grid resource (and how this changes with actual job completion time). With this QoS contract, a parallel job completed by Faucets can have three possible economic outcomes: payoff at soft deadline; a decreased payoff at hard deadline (after soft deadline); and penalty after hard deadline.

Faucets uses the tendering/contract-net economic model. First, it determines the list of Grid resources that are able to satisfy the job's execution requirements. Then, requests are sent out to each of these Grid resources to inform them about this new job. Grid resources can choose to decline or reply with a bid. The user then chooses the Grid resource when all the bids are collected.

Faucets has a producer participant focus and competitive trading environment as each Grid resource aims to maximize its own profit and resource utilization and thus compete with other resources. Faucets considers the time QoS attribute since each Grid resource that receives a new job request first checks that it can satisfy the job's QoS contract before replying with a bid. The cost QoS attribute is decided by the user who then chooses the resource to execute based on the bids of the Grid resources.

Faucets currently uses a centralized management control where the Faucets Server (FS) maintains the list of resources and applications that user can execute. However, the ultimate aim of Faucets is to have a distributed management control to improve scalability. Time-shared scheduling support is employed in Faucets where adaptive jobs are executed simultaneously but on different proportions of the allocated processors. A centralized accounting mechanism at the FS keeps track of participating Grid resources so that owners of these Grid resources can earn credits to execute jobs on other Grid resources. Faucets is primarily designed to support parallel job processing type only where the constraint-based QoS contract of a parallel job is given at job submission and remains static throughout the execution.

In Faucets, the resource allocation domain operates in an internal manner where each Grid resource is only aware of jobs submitted via the FS and not other remote Grid resources. To maximize system utilization at each Grid resource, Faucets allocates a proportional number of processors to jobs based on their QoS priorities since jobs are adaptive to changing number of processors. A new job with a higher priority is allocated a larger proportion of processors, thus resulting in existing jobs becoming entitled to shrinking proportion of processors. This results in soft QoS support. Faucets does not describe how utility-driven performance can be evaluated.

5.6. Nimrod/G

Nimrod/G [28] is the grid-enabled version of Nimrod [78] that allows user to create and execute parameter sweep applications on the Grid. Its design is based on a commodity market economic model where each Nimrod/G broker associated with a user obtains service prices from Grid traders at each different Grid resource location. Nimrod/G supports a consumer participant focus that considers deadline (time) and budget (cost) QoS constraints specified by the user for running his application. Prices of resources thus vary between different executing applications depending on the time and selection of Grid resources that suits the QoS constraints. This means that users have to compete with one another in order to maximize their own personal benefits, thus establishing a competitive trading environment.

Each Nimrod/G broker acts on behalf of its user to discover the best resources for their application and does not communicate with other brokers, thus implementing a decentralized management control. It also has its own decentralized accounting mechanism to ensure that the multiple tasks within the parameter sweep application do not violate the overall constraints. In addition, the Nimrod/G broker is able to operate in a highly dynamic Grid environment where resources are heterogeneous since they are managed by different owners, each having their own operating policies. The broker does not need to know the scheduling support of each Grid resource as each resource feedbacks to the broker their estimated completion time for a task.

A parameter sweep application generates multiple independent tasks with different parameter values that can execute sequentially on a processor. For each parameter sweep application, the Nimrod/G broker creates a plan to assign tasks to resources that either optimizes time or cost within the deadline and budget constraints or only satisfies the constraints without any optimization [79]. The QoS constraints for a parameter sweep application can only be specified before the creation of the plan and remains static when the resource broker discovers and schedules suitable resources.

The Nimrod/G broker discovers external Grid resources across multiple administrative domains. Resources are discovered and assigned progressively for the multiple tasks within an application depending on current resource availability that is beyond the control of the broker. Therefore, Nimrod/G is only able to provide soft QoS support as it tries its best to fulfill the QoS constraints. It supports some level of adaptive resource allocation updates as it attempts to discover resources for remaining tasks yet to be scheduled based on the remaining budget from scheduled tasks so that the overall budget is not exceeded. It also attempts to reschedule tasks to other resources if existing scheduled tasks fails to start execution. However, Nimrod/G stops assigning remaining tasks once the deadline or budget QoS constraint is violated, thus wasting budget and time spent on already completed tasks. Nimrod/G does not describe how utility-driven performance can be evaluated.

5.7. Tycoon

Tycoon [30] examines resource allocation in Grid environments where users are self-interested with unpredictable demands and service hosts are unreliable with changing availability. It implements a two-tier resource allocation architecture that differentiates between user strategy and allocation mechanism. The user strategy captures high-level preferences that are application-dependent and vary across users, while the allocation mechanism provides means to solicit true user valuations for more efficient execution. The separation of user strategy and allocation mechanism therefore allows both requirements not to be limited and dependent of one another.

Each service host utilizes an auction share scheduler that holds First-price auctions to determine resource allocation. The request with the highest bid is then allocated the processor time slice. The bid is computed as the pricing rate that the user pays for the required processor time, hence both time and cost QoS attributes are considered. Consumer participant focus is supported as users can indicate whether the service requests are latency-sensitive or throughput-driven. Based on these preferences, consumers have to compete with one another for Grid sites that can satisfy their service requests.

A host self-manages its local selection of applications, thus maintaining decentralized resource management. Hosts are heterogeneous since they are installed in various administrative domains and owned by different owners. Applications are assigned processor time slices so that multiple requests can be concurrently executed. Each host also keeps accounting information of its local applications to calculate the usage-based service cost to be paid by the user and determine prices of future resource reservation for risk-averse applications.

Tycoon is assumed to handle general service applications that include Web and database services. Service execution requests are specified in terms of constraints such as the amount of cost to spend and the deadline for completion. These constraints do not change after initial specification. Each auction share scheduler performs resource assignment internally within the service host. It also enables adaptive resource allocation updates as new service requests modify and reduce the current resource entitlements of existing executing requests. This results in soft QoS support that can have a negative impact for risk-averse and latency-sensitive applications. To minimize this, Tycoon allows users to reserve resources in advance to ensure sufficient entitlements.

The performance evaluation concentrates on consumer using a user-centric time evaluation factor. A metric called scheduling error assesses whether users get their specified amount of resources and also justifies the overall fairness for all users. The mean latency is also measured for latency-sensitive applications to examine whether their requests are fulfilled. Simulation results show that the Tycoon is able to achieve high fairness and low latency when compared with a simple proportional-share strategy. In addition, Tycoon minimizes interaction protocol overheads by holding auctions internally within each service host to reduce communication across hosts.

5.8. Stanford Peers Initiative

The Stanford Peers Initiative [34] employs a peer-to-peer data trading framework to create a digital archiving system. It utilizes a bid trading auction mechanism where a local site that wants to replicate its collection holds an auction to solicit bids from remote sites by first announcing its request for storage space. Each interested remote site then returns a bid that reflects the amount of disk storage space to request from the local site in return for providing the requested storage space. The local site then selects the remote site with the lowest bid for maximum benefit.

An overall cooperative trading environment with both producer and consumer participant focus is supported through a bartering system whereby sites exchange free storage spaces to benefit both themselves and others. Each site minimizes the cost of trading, which is the amount of disk storage space it has to provide to the remote site for the requested data exchange. The Stanford Peers Initiative implements decentralized management control as each site makes its own decision to select the most suitable remote sites to replicate its data collection.

Each site is external of one another and can belong to different owners. Once a remote site is selected, the specified amount of storage space remains fixed, hence implying non-adaptive resource allocation

Table VII. Comparison of cluster and Grid/peer-to-peer computing.

Characteristic	Cluster	Grid/peer-to-peer
Administrative domain	Single	Multiple
Availability	Committed	Uncommitted
Component	Homogeneous	Heterogeneous
Coupling	Tight	Loose
Location	Local	Global
Ownership	Single	Multiple
Policy	Homogeneous	Heterogeneous
Size	Limited	Unlimited

update. The job model taxonomy does not apply to the Stanford Peers Initiative because the allocation of resources is expressed in terms of data exchange and not jobs.

The Stanford Peers Initiative evaluates performance based on reliability against failures since the focus of an archiving system is to preserve data as long as possible. Reliability is measured using the mean time to failure (MTTF) for each local site that is both a producer and consumer. Simulation results show that sites that use bid trading achieve a higher reliability than sites that trade equal amounts of space without bidding.

6. GAP ANALYSIS

The gap analysis differentiates between what is required and what is currently available for market-based RMSs to support utility-driven cluster computing. It first examines key attributes of utility-driven cluster computing through a comparison with other computing platforms such as Grids and peer-to-peer. Then, it identifies design issues that are still outstanding for constructing practical market-based cluster RMSs to support utility-driven cluster computing.

6.1. Characteristics of utility-driven cluster computing

A comparison between the characteristics of cluster computing and that of other computing platforms enables a better understanding of why market-based cluster RMSs designed for other computing platforms may not be applicable or suitable for cluster computing and *vice versa*. To be consistent with the scope of the survey, the other computing platforms for comparison are again only restricted to Grid and peer-to-peer computing.

Table VII highlights how cluster computing differs from Grid and peer-to-peer computing. It can be seen that a cluster system has different characteristics from a Grid or peer-to-peer system. In particular, a cluster system has simpler characteristics that can be managed more easily.

A cluster system is owned by a single owner and resides in a single administrative domain, while a Grid or peer-to-peer system comprises of resources owned by multiple owners and distributed across multiple administrative domains. It is easier to gain access to cluster resources as a single owner can easily commit the cluster system to be available for use. On the other hand, resources in a Grid or peer-to-peer system are uncommitted and may not always be available as various owners have the authority to decline access at anytime. This high level of uncertainty can thus lead to low reliability if not handled properly.

A cluster system is tightly coupled, located at a single local site and limited in size, whereas a Grid or peer-to-peer system is loosely coupled with multiple remote sites distributed globally and can be unlimited in size. This implies that a cluster system does not need to be as scalable as a Grid or peer-to-peer system. In a cluster system, nodes typically have homogeneous components and policies to facilitate standard management and control. On the other hand, each site in a Grid or peer-to-peer system can have heterogeneous components and policies, thus requiring greater coordination effort.

In a cluster system, middlewares such as the cluster RMS creates a Single System Image (SSI) [4] to operate like a single computer that hides the existence of multiple cluster nodes from consumers and manages job execution from a single user interface. The cluster RMS has complete state information of all jobs and resources within the cluster system. To support utility-driven cluster computing, the market-based cluster RMS emphasizes on providing utility from the system perspective collectively for consumers using the cluster system. It can also administer that overall resource allocation to different users is fair, based on criteria such as their SLA requirements in order to maximize aggregate utility for consumers.

On the other hand, a Grid or peer-to-peer system operates like a collection of computing services. In a Grid or peer-to-peer system, each site competes with other sites whereby sites are under different ownership and have heterogeneous policies implemented by various producers (owners). In fact, each Grid or peer-to-peer site can be a cluster system. Thus, the market-based RMS for each site is greedy and only maximizes utility for its individual site, without considering other sites since these sites are beyond its control. To support the heterogeneity of components and policies across sites, the market-based RMS at each site may need to communicate using multiple interfaces, which is a tedious process. Moreover, the market-based RMS needs to consider scalability issues, such as communication and data transfer costs owing to sites being distributed globally over multiple administrative domains.

Therefore, market-based RMSs designed to achieve utility in Grid or peer-to-peer computing are not suitable for cluster computing since both computing platforms have distinctive characteristics and varying emphasis of utility. In addition, market-based RMSs designed for Grid or peer-to-peer systems address complexities that do not exist and are thus redundant in cluster systems.

6.2. Outstanding issues

From the survey, we can identify outstanding issues that are yet to be explored but are important for market-based cluster RMSs to support utility-driven cluster computing in practice. There are currently only a few market-based RMSs designed specifically for cluster systems such as Cluster-On-Demand [25], Enhanced MOSIX [46], Libra [24] and REXEC [23]. Among them, only Libra and REXEC provide a consumer participant focus that is crucial for satisfying QoS-based requests to generate utility for consumers. None of these market-based cluster RMSs supports other important QoS attributes such as reliability and trust/security that are essential in a utility-driven service market where consumers pay for usage and expect good quality service.

These market-based cluster RMSs mostly only support fairly simple job models with sequential job processing type and single-task job composition and static QoS update. However, more advanced job models that comprise parallel job processing type and multiple-task job composition, such as message-passing, parameter-sweep and workflow applications are typically required by users for their work execution in reality. Consumers may also need to modify their initial QoS specifications after job submission as their service requirements may vary over time, thus requiring the support of dynamic QoS updates.

A larger pool of resources can be available for usage if the market-based cluster RMSs can be extended to discover and utilize external resources in other cluster systems or Grids. Service requests may then continue to be fulfilled when there are insufficient internal resources within the cluster systems to meet demands. Current market-based cluster RMSs also do not support both soft and hard QoS supports.

For evaluation purposes, both system-centric and user-centric evaluation factors need to be defined to measure the effectiveness of market-based cluster RMSs in achieving overall system performance and actual benefits for both consumers and producers. Metrics for measuring system and interaction protocol overheads incurred by the market-based cluster RMSs are also required to evaluate their efficiency.

There is also a possibility of incorporating strengths proposed in market-based RMSs for other computing platforms in the context of cluster computing. For instance, the tendering/contract-net economic model in Faucets [29] may be applied in a cluster system with decentralized management control where the consumer determines the resource selection by choosing the best node based on bids from competing cluster nodes. Optimization-based QoS specification in Nimrod/G [28] and the ‘auction share’ scheduling algorithm in Tycoon [30] can improve utility for consumers, in particular those with latency-sensitive applications. Bartering concepts in the Stanford Peers Initiative [34] can augment the level of sharing across internal and external resource allocation domains.

7. SUMMARY AND CONCLUSION

In this paper, a taxonomy is proposed to characterize and categorize various market-based cluster RMSs that can support utility-driven cluster computing in practice. The taxonomy emphasizes on five different perspectives: (i) the Market Model; (ii) the Resource Model; (iii) the Job Model; (iv) the Resource Allocation Model; and (v) the Evaluation Model. A survey is also conducted where the taxonomy is mapped to selected market-based RMSs designed for both cluster and other computing platforms. The survey enables us to analyze the gap between what is already available in existing market-based cluster RMSs and what is still required so that we can identify outstanding research issues that can result in more practical market-based cluster RMSs being designed in future. The mapping of the taxonomy to the range of market-based RMSs has successfully demonstrated that the proposed taxonomy is sufficiently comprehensive to characterize existing and future market-based RMSs for utility-driven cluster computing.

ACKNOWLEDGEMENTS

We would like to acknowledge all developers of the market-based RMSs described in the paper. This work is partially supported by the Australian Research Council (ARC) Discovery Project and StorageTek Fellowship for Grid Computing.

REFERENCES

1. Gropp W, Lusk E, Sterling T (eds.). *Beowulf Cluster Computing with Linux* (2nd edn). MIT Press: Cambridge, MA, 2003.
2. Pfister GF. *In Search of Clusters* (2nd edn). Prentice-Hall PTR: Upper Saddle River, NJ, 1998.
3. Buyya R (ed.). *High Performance Cluster Computing: Architectures and Systems*. Prentice-Hall PTR: Upper Saddle River, NJ, 1999.
4. Buyya R, Cortes T, Jin H. Single system image. *The International Journal of High Performance Computing Applications* 2001; **15**(2):124–135.
5. University of Wisconsin-Madison. Condor version 6.7.1 manual. http://www.cs.wisc.edu/condor/manual/v6.7 [6 December 2004].
6. IBM. LoadLeveler for AIX 5L version 3.2 using and administering, October 2003. http://www-1.ibm.com/servers/eserver/pseries/library/sp_books/loadleveler.html [6 December 2004].
7. Platform Computing. LSF version 4.1 administrator's guide, 2001. http://www.platform.com/services/support [6 December 2004].
8. Altair Grid Technologies. OpenPBS release 2.3 administrator guide, August 2000. http://www.openpbs.org/docs.html [6 December 2004].
9. Sun Microsystems. Sun ONE Grid Engine, administration and user's guide, October 2002. http://gridengine.sunsource.net/project/gridengine/documentation.html [6 December 2004].
10. Foster I, Kesselman C (eds.). *The Grid 2: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann: San Francisco, CA, 2003.
11. Buyya R, Abramson D, Giddy J. A case for economy Grid architecture for service oriented Grid computing. *Proceedings of the 10th International Heterogeneous Computing Workshop (HCW 2001)*, San Francisco, CA, April 2001. IEEE Computer Society Press: Los Alamitos, CA, 2001.
12. Buyya R, Abramson D, Giddy J. Nimrod/G: An architecture for a resource management and scheduling system in a global Computational Grid. *Proceedings of the 4th International Conference on High Performance Computing in Asia-Pacific Region (HPC Asia 2000)*, Beijing, China, May 2000. IEEE Computer Society Press: Los Alamitos, CA, 2000.
13. Venugopal S, Buyya R, Winton L. A Grid service broker for scheduling distributed data-oriented applications on global Grids. *Proceedings of the 2nd International Workshop on Middleware for Grid Computing (MGC 2004)*, Toronto, Canada, October 2004. ACM Press: New York, NY, 2004; 75–80.
14. Yu J, Buyya R. A novel architecture for realizing Grid workflow using Tuple Spaces. *Proceedings of the 5th International Workshop on Grid Computing (GRID 2004)*, Pittsburgh, PA, November 2004. IEEE Computer Society Press: Los Alamitos, CA, 2004; 119–128.
15. The TeraGrid project. http://www.teragrid.org [6 December 2004].
16. The LHC computing Grid project. http://lcg.web.cern.ch/LCG [6 December 2004].
17. The NAREGI project. http://www.naregi.org [6 December 2004].
18. The APAC Grid project. http://www.apac.edu.au [6 December 2004].
19. IBM Grid computing. http://www.ibm.com/grid [6 December 2004].
20. HP Grid computing. http://www.hp.com/techservers/grid [6 December 2004].
21. Sun Microsystems utility computing. http://www.sun.com/service/utility [6 December 2004].
22. Buyya R, Abramson D, Venugopal S. The Grid economy. *Proceedings of the IEEE* 2005; **93**(3):698–714.
23. Chun BN, Culler DE. Market-based proportional resource sharing for clusters. *Technical Report CSD-1092*, Computer Science Division, University of California at Berkeley, January 2000.
24. Sherwani J, Ali N, Lotia N, Hayat Z, Buyya R. Libra: A computational economy-based job scheduling system for clusters. *Software: Practice and Experience* 2004; **34**(6):573–590.
25. Irwin DE, Grit LE, Chase JS. Balancing risk and reward in a market-based task service. *Proceedings of the 13th International Symposium on High Performance Distributed Computing (HPDC13)*, Honolulu, HI, June 2004. IEEE Computer Society Press: Los Alamitos, CA, 2004; 160–169.
26. Anastasiadi A, Kapidakis S, Nikolaou C, Sairamesh J. A computational economy for dynamic load balancing and data replication. *Proceedings of the 1st International Conference on Information and Computation Economies (ICE '98)*, Charleston, SC, October 1998. ACM Press: New York, NY, 1998; 166–180.
27. Stonebraker M, Devine R, Kornacker M, Litwin W, Pfeffer A, Sah A, Staelin C. An economic paradigm for query processing and data migration in Mariposa. *Proceedings of the 3rd International Conference on Parallel and Distributed Information Systems (PDIS '94)*, Austin, TX, September 1994. IEEE Computer Society Press: Los Alamitos, CA, 1994; 58–68.
28. Abramson D, Buyya R, Giddy J. A computational economy for Grid computing and its implementation in the Nimrod-G resource broker. *Future Generation Computer Systems* 2002; **18**(8):1061–1074.
29. Kalé LV, Kumar S, Potnuru M, DeSouza J, Bandhakavi S. Faucets: Efficient resource allocation on the Computational Grid. *Proceedings of the International Conference on Parallel Processing (ICPP 2004)*, Montreal, Canada, August 2004. IEEE Computer Society Press: Los Alamitos, CA, 2004; 396–405.

30. Lai K, Huberman BA, Fine L. Tycoon: A distributed market-based resource allocation system. *Technical Report cs.DC/0404013*, HP Lab, Palo Alto, April 2004.
31. Malone TW, Fikes RE, Grant KR, Howard MT. Enterprise: A market-like task scheduler for distributed computing environments. *The Ecology of Computation*, Huberman BA (ed.). Elsevier Science Publisher: New York, NY, 1988; 177–205.
32. Waldspurger CA, Hogg T, Huberman BA, Kephart JO, Stornetta WS. Spawn: A distributed computational economy. *IEEE Transactions on Software Engineering* 1992; **18**(2):103–117.
33. Bredin J, Kotz D, Rus D. Utility driven mobile-agent scheduling. *Technical Report PCS-TR98-331*, Department of Computer Science, Dartmouth College, October 1998.
34. Cooper BF, Garcia-Molina H. Bidding for storage space in a peer-to-peer data preservation system. *Proceedings of the 22nd International Conference on Distributed Computing Systems (ICDCS 2002)*, Vienna, Austria, July 2002. IEEE Computer Society Press: Los Alamitos, CA, 2002; 372–381.
35. Alexandrov AD, Ibel M, Schauser KE, Scheiman CJ. SuperWeb: Research issues in Java-based global computing. *Concurrency: Practice and Experience* 1997; **9**(6):535–553.
36. Regev O, Nisan N. The POPCORN market—an online market for computational resources. *Proceedings of the 1st International Conference on Information and Computation Economies (ICE '98)*, Charleston, SC, October 1998. ACM Press: New York, NY, 1998; 148–157.
37. Lalis S, Karipidis A. JaWS: An open market-based framework for distributed computing over the Internet. *Proceedings of the 1st International Workshop on Grid Computing (GRID 2000)*, Bangalore, India, December 2000 (*Lecture Notes in Computer Science*, vol. 1971). Springer: Heidelberg, Germany, 2000; 36–46.
38. Casavant TL, Kuhl JG. A taxonomy of scheduling in general-purpose distributed computing systems. *IEEE Transactions on Software Engineering* 1988; **14**(2):141–154.
39. Rotithor HG. Taxonomy of dynamic task scheduling schemes in distributed computing systems. *IEEE Proceedings of Computers and Digital Techniques* 1994; **141**(1):1–10.
40. Ekmecić I, Tartalja I, Milutinović V. EM^3: A taxonomy of heterogeneous computing systems. *IEEE Computer* 1995; **28**(12):68–70.
41. Ekmecić I, Tartalja I, Milutinović V. A survey of heterogeneous computing: Concepts and systems. *Proceedings of the IEEE* 1996; **84**(8):1127–1144.
42. Braun TD, Siegel HJ, Beck N, Bölöni L, Maheswaran M, Reuther AI, Robertson JP, Theys MD, Yao B. A taxonomy for describing matching and scheduling heuristics for mixed-machine heterogeneous computing systems. *Proceedings of the 17th Symposium on Reliable Distributed Systems*, West Lafayette, IN, October 1998. IEEE Computer Society Press: Los Alamitos, CA, 1998; 330–335.
43. Krauter K, Buyya R, Maheswaran M. A taxonomy and survey of Grid resource management systems for distributed computing. *Software: Practice and Experience* 2002; **32**(2):135–164.
44. Buyya R, Abramson D, Giddy J, Stockinger H. Economic models for resource management and scheduling in Grid computing. *Concurrency and Computation: Practice and Experience* 2002; **14**(13–15):1507–1542.
45. Yeo CS, Buyya R. Service level agreement based allocation of cluster resources: Handling penalty to enhance utility. *Proceedings of the 7th IEEE International Conference on Cluster Computing (CLUSTER 2005)*, Boston, MA, September 2005. IEEE Computer Society Press: Los Alamitos, CA, 2005 (CDROM).
46. Amir Y, Awerbuch B, Barak A, Borgstrom RS, Keren A. An opportunity cost approach for job assignment in a scalable computing cluster. *IEEE Transactions on Parallel and Distributed Systems* 2000; **11**(7):760–768.
47. Ranjan R, Buyya R. A case for cooperative and incentive-based coupling of distributed clusters. *Proceedings of the 7th IEEE International Conference on Cluster Computing (CLUSTER 2005)*, Boston, MA, September 2005. IEEE Computer Society Press: Los Alamitos, CA, 2005 (CDROM).
48. Barmouta A, Buyya R. GridBank: A Grid accounting services architecture (GASA) for distributed systems sharing and integration. *Proceedings of the 3rd Workshop on Internet Computing and E-Commerce (ICEC 2003), International Parallel and Distributed Processing Symposium (IPDPS 2003)*, Nice, France, April 2003. IEEE Computer Society Press: Los Alamitos, CA, 2003.
49. Jackson SM. Allocation management with QBank. http://www.emsl.pnl.gov/docs/mscf/qbank [6 December 2004].
50. IBM. Business Process Execution Language for Web Services, version 1.1. http://www.ibm.com/developerworks/library/specification/ws-bpel [25 July 2005].
51. Global Grid Forum. Web Services Agreement Specification (WS-Agreement), version 1.1, draft 18. http://www.gridforum.org/Meetings/GGF11/Documents/draft-ggf-graap-agreement.pdf [25 July 2005].
52. Keller A, Ludwig H. The WSLA framework: Specifying and monitoring service level agreements for Web services. *Journal of Network and Systems Management* 2003; **11**(1):57–81.
53. Yeo CS, Buyya R. Pricing for utility-driven resource management and allocation in clusters. *Proceedings of the 12th International Conference on Advanced Computing and Communication (ADCOM 2004)*, Ahmedabad, India, December 2004. Allied Publisher: New Delhi, India, 2004; 32–41.

54. Byde A, Sallé M, Bartolini C. Market-based resource allocation for utility data centers. *Technical Report HPL-2003-188*, HP Lab, Bristol, September 2003.
55. AuYoung A, Chun BN, Snoeren AC, Vahdat A. Resource allocation in federated distributed computing infrastructures. *Proceedings of the 1st Workshop on Operating System and Architectural Support for the on demand IT InfraStructure (OASIS 2004)*, Boston, MA, October 2004 (CDROM).
56. Eymann T, Reinicke M, Ardaiz O, Artigas P, Freitag F, Navarro L. Decentralized resource allocation in application layer networks. *Proceedings of the 3rd International Symposium on Cluster Computing and the Grid (CCGrid 2003)*, Tokyo, Japan, May 2003. IEEE Computer Society Press: Los Alamitos, CA, 2003; 645–650.
57. Wolski R, Plank JS, Brevik J, Bryan T. Analyzing market-based resource allocation strategies for the Computational Grid. *International Journal of High Performance Computing Applications* 2001; **15**(3):258–281.
58. Chen M, Yang G, Liu X. Gridmarket: A practical, efficient market balancing resource for Grid and P2P computing. *Proceedings of the 2nd International Workshop on Grid and Cooperative Computing (GCC 2003)*, Shanghai, China, December 2003 (*Lecture Notes in Computer Science*, vol. 3033). Springer: Heidelberg, Germany, 2003; 612–619.
59. Grosu D, Das A. Auction-based resource allocation protocols in Grids. *Proceedings of the 16th International Conference on Parallel and Distributed Computing and Systems (PDCS 2004)*, Cambridge, MA, November 2004. ACTA Press: Calgary, Canada, 2004; 20–27.
60. Chen C, Maheswaran M, Toulouse M. Supporting co-allocation in an auctioning-based resource allocator for Grid systems. *Proceedings of the 11th International Heterogeneous Computing Workshop (HCW 2002)*, Fort Lauderdale, FL, April 2002. IEEE Computer Society Press: Los Alamitos, CA, 2002.
61. Padala P, Harrison C, Pelfort N, Jansen E, Frank MP, Chokkareddy C. OCEAN: The open computation exchange and arbitration network, a market approach to meta computing. *Proceedings of the 2nd International Symposium on Parallel and Distributed Computing (ISPDC 2003)*, Ljubljana, Slovenia, October 2003. IEEE Computer Society Press: Los Alamitos, CA, 2003; 185–192.
62. Miller MS, Drexler KE. Incentive engineering for computational resource management. *The Ecology of Computation*, Huberman BA (ed.). Elsevier Science Publisher: New York, NY, 1988; 231–266.
63. Backschat M, Pfaffinger A, Zenger C. Economic-based dynamic load distribution in large workstation networks. *Proceedings of the 2nd International Euro-Par Conference (Euro-Par 1996)*, Lyon, France, August 1996 (*Lecture Notes in Computer Science*, vol. 1124). Springer: Heidelberg, Germany, 1996; 631–634.
64. Ferguson DF, Yemini Y, Nikolaou C. Microeconomic algorithms for load balancing in distributed computer systems. *Proceedings of the 8th International Conference on Distributed Computing Systems (ICDCS'88)*, San Jose, CA, June 1988. IEEE Computer Society Press: Los Alamitos, CA, 1988; 491–499.
65. Kurose JF, Simha R. A microeconomic approach to optimal resource allocation in distributed computer systems. *IEEE Transactions on Computers* 1989; **38**(5):705–717.
66. Yemini Y, Dailianas A, Florissi D, Huberman G. MarketNet: Protecting access to information systems through financial market controls. *Decision Support Systems* 2000; **25**(1–2):205–216.
67. Preist C, Byde A, Bartolini C. Economic dynamics of agents in multiple auctions. *Proceedings of the 5th International Conference on Autonomous Agents (AGENTS 2001)*, Montreal, Canada, May–June 2001. ACM Press: New York, NY, 2001; 545–551.
68. Stoica I, Abdel-Wahab H, Pothen A. A microeconomic scheduler for parallel computers. *Proceedings of the 1st Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP '95)*, Santa Barbara, CA, April 1995 (*Lecture Notes in Computer Science*, vol. 949). Springer: Heidelberg, Germany, 1995; 200–218.
69. Wellman MP. A market-oriented programming environment and its application to distributed multicommodity flow problems. *Journal of Artificial Intelligence Research* 1993; **1**:1–23.
70. Amir Y, Awerbuch B, Borgstrom RS. A cost-benefit framework for online management of a metacomputing system. *Proceedings of the 1st International Conference on Information and Computation Economies (ICE '98)*, Charleston, SC, October 1998. ACM Press: New York, NY, 1998; 140–147.
71. Reed D, Pratt I, Menage P, Early S, Stratford N. Xenoservers: Accountable execution of untrusted programs. *Proceedings of the 7th Workshop on Hot Topics in Operating Systems (HotOS-VII)*, Rio Rico, AZ, March 1999. IEEE Computer Society Press: Los Alamitos, CA, 1999; 136–141.
72. Chase JS, Irwin DE, Grit LE, Moore JD, Sprenkle SE. Dynamic virtual clusters in a Grid site manager. *Proceedings of the 12th International Symposium on High Performance Distributed Computing (HPDC12)*, Seattle, WA, June 2003. IEEE Computer Society Press: Los Alamitos, CA, 2003; 90–100.
73. Barak A, La'adan O. The MOSIX multicomputer operating system for high performance cluster computing. *Future Generation Computer Systems* 1998; **13**(4–5):361–372.
74. Chun BN, Culler DE. User-centric performance analysis of market-based cluster batch schedulers. *Proceedings of the 2nd International Symposium on Cluster Computing and the Grid (CCGrid 2002)*, Berlin, Germany, May 2002. IEEE Computer Society Press: Los Alamitos, CA, 2002; 22–30.

75. Kalé LV, Krishnan S. Charm++: Parallel programming with message-driven objects. *Parallel Programming Using C++*, Wilson GV, Lu P (eds.). MIT Press: Cambridge, MA, 1996; 175–213.
76. Bhandarkar M, Kalé LV, de Sturler E, Hoeflinger J. Adaptive load balancing for MPI programs. *Proceedings of the International Conference on Computational Science (ICCS 2001)*, San Francisco, CA, May 2001 (*Lecture Notes in Computer Science*, vol. 2074). Springer: Heidelberg, Germany, 2001; 108–117.
77. Kalé LV, Kumar S, DeSouza J. A malleable-job system for timeshared parallel machines. *Proceedings of the 2nd International Symposium on Cluster Computing and the Grid (CCGrid 2002)*, Berlin, Germany, May 2002. IEEE Computer Society Press: Los Alamitos, CA, 2002; 215–222.
78. Abramson D, Sosic R, Giddy J, Hall B. Nimrod: A tool for performing parametrised simulations using distributed workstations. *Proceedings of the 4th International Symposium on High Performance Distributed Computing (HPDC4)*, Pentagon City, VA, August 1995. IEEE Computer Society Press: Los Alamitos, CA, 1995; 112–121.
79. Buyya R, Giddy J, Abramson D. An evaluation of economy-based resource trading and scheduling on computational power Grids for parameter sweep applications. *Proceedings of the 2nd Annual Workshop on Active Middleware Services (AMS 2000)*, Pittsburgh, PA, August 2000. Kluwer Academic Publishers: Dordrecht, Netherlands, 2000.