Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 50 (2015) 653 – 662

Procedia Computer Science

2nd International Symposium on Big Data and Cloud Computing (ISBCC'15)

A CONTEXT-AWARE TASK SCHEDULING IN AD-HOC DESKTOP GRIDS

Neelanarayanan Venkataraman.

School of Computing Science and Engineering, VIT University, Chennai.India-600127, India
IT University of Copenhagen, Rued Langgaards Vej 7, DK-2300, Copenhagen S, Denmark

Abstract

To harvest idle, unused computational resources in networked environments, researchers have proposed different architectures for desktop grid infrastructure. In this paper, we present one such infrastructure, called the Mini-Grid Framework for resource management in ad hoc grids using market-based scheduling and context-based resource and application modeling. The framework proposes peer-to-peer architecture that supports several futures: decentralized task distribution, small scale ad hoc grid formation, and symmetric resource. Furthermore, users can model and specify non-performance based parameters that influence resource allocation. Different types of resources can provide similar capabilities but with varying degrees of quality of service. Hence, the resource capabilities are required to be presented in such a way that resource providers can evaluate their capabilities against the requested capabilities for task execution, and the resource consumers can find optimal resources.

Peer-review under responsibility of scientific committee of 2nd International Symposium on Big Data and Cloud Computing (ISBCC'15)

Keywords: context information management, desktop grid computing, resource description,quality of service

1. Introduction

Let us consider a typical environment of the computer set-up in a biological research-based institution. Usually, Biologists working with laboratory-based and theoretical molecular biological research have access to one or two clusters and have to share their computation time with others. At the same time, these biologists work in labs or institutions, which have a large number of desktops and laptops. Those machines are usually under-utilized and are only available to individual users. Organizing such desktop machines as a distributed system for computations or other kinds of resource sharing is highly desirable. For example, a group of scientists or researchers can contribute their desktop machines when idle for a common objective. In this example, every participant wants to participate for a limited amount of time, normally until the participant has some interest in the collaboration. For such a group it

Peer-review under responsibility of scientific committee of 2nd International Symposium on Big Data and Cloud Computing (ISBCC'15)
doi:10.1016/j.procs.2015.04.099

may be infeasible to build or participate in a classical grid infrastructure or volunteer computing infrastructure because of administrative overheads, cost, etc. Thus support for formation of "ad hoc" grid infrastructure among participants of one-time or short lived collaboration is required. Furthermore, such an infrastructure is suitable to applications with low communication to computation ratio, such as parallel search algorithms. An ad-hoc grid is a distributed system with the aim of harnessing unused computational resources inside an organization. It can be considered as a temporary coalition of resources contributed by individuals to achieve common objective of access to high computational environment. In an ad hoc grid, every resource can spontaneously arise as a resource consumer or a resource producer at any time when it needs a resource or it possesses an idle resource.

1.1 Grid Computing

Many efforts to parallelize RNA structure prediction algorithms have been directed towards supercomputer clusters [9, 10] or even specialized multithreaded hardware [12]. However, installing, configuring and customizing such solutions require high technical knowledge and dedicated hardware and software resources. For these reasons, the deployment and operational costs of such systems are substantial, which prevents their adoption and direct use by biological researchers. Another option has been provided by volunteer computing systems, which allow the formation of parallel computing networks by enabling ordinary internet users to share their computer's idle processing power [8, 1]. One example is the Folding@Home project [4]. However, such volunteer computing systems require a centralized control system responsible for managing the contributing clients, who in turn periodically request work from a central server. As such, volunteer computing is highly asymmetric; it is a 'master-slave' architecture in which volunteers supply computing resource but do not submit any work to be done. Public outreach and incentive structures (like high-score competitions) play a significant role in attracting volunteers.

1.2 Bag-of-Task Applications

Bioinformatics applications are computationally very intensive and require vast amounts of processing power and memory requirements. For example, finding genes in DNA sequences, predicting the structure and functions of new proteins, clustering proteins into families, aligning similar proteins, and building phylogenetic tree showing the evolutionary relationships are all computationally intensive. Such kind of computational biology problems require design and development of solutions that maximize performance and programmability. Using computational Grid infrastructure is an effective way to tackle the problem. The computational Grid infrastructures are built using a set of processors that are able to cooperate in solving computational problems. This cooperation is achieved by splitting the computational load of the problem into parts, and then combining the partial computations to obtain the result. Thus modeling a bioinformatics application as a Bag-of-Task (BoT) application, where each computational process (task) works at its own rhythm in an asynchronous fashion with complete independence from other computational process, makes it possible to use computational Grid infrastructure to meet their computational demands.

1.3 Resource and Application Modelling

In Grid, resource provisioning can be implemented using either "resource-push" or "resource-pull" model. In resource push model, a "resource manager" or "scheduler" selects the best resource provider to run the tasks on behalf of a resource consumer, for example, the condor match maker [14]. On the other hand, in the resource-pull model, resource providers periodically request the tasks from the resource manager or scheduler, for example, the BOINC scheduling server [4]. In the resource-push model, to make appropriate decisions, schedulers need accurate information in real time about resource providers of interest. This information can be made available using either "information-push" or "information-pull" model. In the information-push model, information is send periodically (adopted, for instance, by the Globus Toolkit MDS [20]). On the other hand in the information-pull model, information is collected from resource providers on demand (adopted, for instance, by the Legion resource management system [8]). The information-push model may cause schedulers to use stale (i.e. inaccurate) information.

Different types of resources can provide similar capabilities but with varying degrees of quality of service. Hence, the resource capabilities are required to be presented in such a way that resource providers can evaluate their capabilities against the requested capabilities for task execution, and the resource consumers can find optimal resources. A powerful discovery mechanism can be built if resource's capabilities and requirements are described explicitly, precisely, and unambiguously. Resource consumers need a utility model, representing their resource demand and preferences. And resource providers need an expressive resource description mechanism. To harvest idle, unused computational resources in networked environments, researchers have proposed different architectures for desktop grid infrastructure. However, most of the existing research work focus on centralized approach. In this paper, we present one such infrastructure, called the Mini-Grid Framework for resource management in ad hoc grids using market-based scheduling and context-based resource and application modelling.

2. Related Work

Most of the current approaches to desktop grid computing is very centralized both technically and in use. However, there are situation that requires a transient, short lived and one-time collaboration among participants. Peer-to-peer architecture presents an alternative way to build desktop grid computing systems. Peer-to-peer based desktop grid computing systems: i) are independent from any central infrastructure and ii) permits symmetric use of computational capabilities. However, owing to lack of a centralized server, requires new mechanisms to compensate the server, for example decentralized resource discovery process. Self-organization solves the issues related to abolishing central server. Self-organization is the process where the organization of a system spontaneously increases without being managed by an outside source or central authority [22]. Resources in Grid could self-organize them self by using market based models such as auctions [35]. An economy based resource management helps in building a large scale computational grid; distributes decision-making process; provides a fair basis for access to grid resources; enables both consumers and producers to maximize their utility; helps in regulating the demand and supply; and helps in developing user centric as well as system centric scheduling policies [5].

Market-based resource allocation mechanisms are attractive than traditional mechanisms, such as first-come-first served allocation, resource reservation mechanism or priority queues for the following reasons: i) supports self-organization of resources, ii) eliminate the need for a central component for scheduling decisions, and iii) permits many auctioneers to function in a distributed environment. The use of market concepts for resource management is not new. Different research projects [3, 19, 25, 27, 30] used market concepts for resource management in computational clusters. Market-based resource allocation model which are based on trading and resource brokering policies between resource providers and resource consumers have been proposed in [1, 31, 5, 15, 18]. The mechanism proposed in market-based grid resource allocation include models such as: Auctions (based on outcome of a bidding process), Commodity markets (resource providers specify the charges and resource consumers pay according to the usage of resource), Tenders (based on contracting mechanism that governs the agreement between resource providers and resource consumers), and Posted price (similar to commodity market except that there are special offers from time to time to attract more users) [5]. In addition to these economic models, different pricing schemes apply to the markets such as: Flat price model (fixed price for a period of time), Competitive economic models (dynamic pricing scheme), Usage timing (peak-o-peak - like telephone services), Prediction-based (based on the ability to predict responses by competitors), Loyalty-based (special prices for regular and loyal users), and Advanced contract based (contract established before resource use determines the price) [5].

The objective of market-based resource allocation model is the provision of resources to satisfy demand. A critical step in designing such a market is to characterize the product that will be traded. Thus, from grid resource allocation context, a means to describe desired capabilities by resource consumer and to describe the offered capabilities by resource provider is required.

3. The Mini Grid Framework

The use of computers for data analysis and visualization of results are currently playing a fundamental role in the working methodology of many research groups. As a consequence, having access to high-performance computing has become essential for many applications. Classical grid systems follow either centralized or hierarchical

architecture or a regulated control for membership and access privileges. These grid architectures assume a dedicated administrative authority for policy enforcement, monitoring and access control. There exist well defined collaborations based on some predefined usage policies and access privileges [9].

However, there are situation that requires a transient, short lived and one-time collaboration among participants. One example of such a situation is the scenario where a group of researchers want to contribute their computational resources for a common objective. In this example, every participant wants to participate for a limited amount time, normally until the participant has some interest in the collaboration. For such a group it may be infeasible to build or participate in a computational grid infrastructure because of administrative overheads, cost, etc. In other words, existing grid models do not support the "ad hoc" collaboration among participants. Current volunteer grid architectures though support a wide range of applications with diversity related to scope and requirements they fail to support sporadic collaborations in the absence of a central regulating authority which presents the need for an ad hoc architecture which will promote structural independence, technological independence and control independence for more than a single administrator enabling promotion of better user collaboration [2]. We build a light-weight extensible framework, called the Mini-Grid Framework supporting the formation of peer-to-peer desktop grids.

In this section, we present the Mini-Grid Framework [32, 33], which is the programming API and runtime infrastructure for formation of ad-hoc "mini-grids" in a local area network. The main goal of the proposed infrastructure, based on a peer-to-peer architecture, is to manage a pool of resources; and owing to the infrastructure with applications having easy access to them.

3.1. Conceptual Architecture

A conceptual illustration of the Mini-Grid Framework is shown in Figure 3.1. The Mini-Grid Framework consists of four logical components: *ResourceProviders, ResourceConsumers, TaskBus* and *MessengerComponent*. Each resource participating in Mini-Grid has a software entity called "client". On the one hand, a client can be used by a resource to consume the computational capabilities of other resources. On the other hand, it can also be used by a resource to provide its computational capabilities to other resources. That is, a client can play both "Resource Consumer" and "Resource Provider" roles. The clients in Mini-Grid communicate with each other using the messenger component. The computational tasks are exchanged between the clients using the task bus.

Each client participating in the Mini-Grid environment has three sub-systems namely the task scheduling sub-system, the context awareness sub-system and messaging sub-system. The task scheduling sub-system is responsible for handling issues related to task submission, distribution and execution. The context awareness sub-system is responsible for describing the context of tasks and for collecting and providing context information of resources. The messaging sub-system is responsible for sending and receiving Mini-Grid messages according to the Mini-Grid Messaging Protocol.

As a distributed computing platform, the framework allows Mini-Grid application running on a client acting as resource consumer to submit tasks to the infrastructure which will be executed on clients acting as resource provider. Clients playing resource consumer role accepts tasks coming from Mini-Grid applications, distributes the tasks to resource providers according to a scheduling policy based on auction, transfers application code to executors, collects and stores task results and delivers task results to Mini-Grid application upon request. Client playing resource provider role listens for task announcements, participates in auction process, executes allocated task and returns completed task along with results.

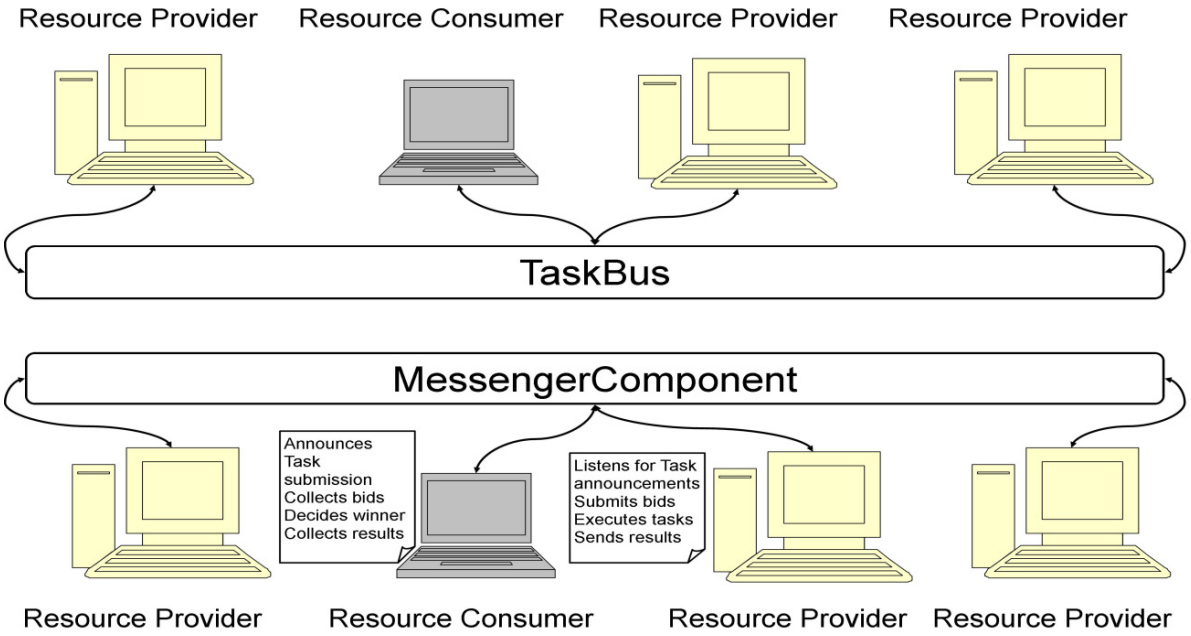

Fig. 3.1. A conceptual model of the Mini-Grid Architecture including the TasksBus, MessengerComponent, ResourceConsumers and Resource Providers.

3.2. Task Scheduling Model

The task scheduling model includes: i) model that describes resource requirement, ii) model that describe available resource, iii) task distribution protocol, and iv) model that describe quality of service requirements of the application. A sequential application can be grid enabled by identifying the independent computations that can be executed concurrently. One way is task decomposition, in which the computations are a set of independent tasks that can be executed in any order. Another way is data decomposition, in which the focus is on the data rather than on the different operations applied to the data. The data is partitioned into certain number of parts and the operation can be performed on the different parts [23]. There are other patterns of computation do exist. However, the above two decomposition are most common. The grid enabled application consists of a collection of independent task with a quality of service requirement. Certain task may require large amount of data to be transferred to or from a remote node before or after the execution of the task at the remote node. However, for better performance each task in the application should have a higher computational complexity index, a ratio between the execution time of the task to its data transfer time.

Each entity in Mini-Grid, for example grid enabled application, task and resource may have a set of characteristics or properties or features or properties. Such information can be called context of the entity. Context models can be used for storing and querying the context information associated with an entity and the relationship among entities. For example, task context model can describe the resource requirements (i.e., execution environment of the task). Further, application context model can describe the quality of service requirements and resource context model can be used to describe the capabilities of a resource.

3.3. Resource Discovery and Selection

Generally, task distribution in volunteer computing environment involves a “pull” or “push” model. In pull

model, the workers (resource providers) request tasks from the master (resource consumer). On the other hand, in push model the master (resource consumer) distributes tasks to arbitrary workers. In the push model, the first task of the scheduler (on behalf of resource consumer) is resource discovery, i.e. to identify a list of available resources. Once the list of possible resources is known, the second task is selecting those resources that most suitable based on constraints imposed by the grid enabled application or the user. Normally, a scheduling policy selects the suitable resources. The scheduling policy determines how to select appropriate resources. It can be classified into three categories: simple, model-based, and heuristics-based [10]. In the simple approach, resources are selected by using First Come First Served (FCFS) or randomly. Further, the model-based approach can be categorized into deterministic, economy, and probabilistic models. The deterministic model is based on structure or topology such as queue, stack, tree, or ring. Resources are deterministically selected according to the properties of structure or topology. For example, in a tree topology, tasks are allocated from parent nodes to child nodes. In the economy model, scheduling decision is based on market economy (i.e., price and budget). In the probabilistic model, resources are selected in probabilistic manners (such as Markov model, machine learning or genetic algorithm). In the heuristic-based approach resources are selected by ranking, matching, and exclusion methods on the basis of performance, capability, weight, precedence, workload, availability, location, reputation, trust, etc. The matching method chooses the most suitable resources in accordance to evaluation function. The exclusion method excludes resource according to criteria, and then chooses the most appropriate one among the survivors. Ranking, matching and exclusion methods can be used together or separately. Workload, availability, location, time zone, reputation, trust, performance, capability and weight are used as criteria for selecting resources [10].

The use of market concepts for resource management is not new. Different research projects [3, 19, 25, 27, 30] used market concepts for resource management in computational clusters. Market-based resource allocation model which are based on trading and resource brokering policies between resource providers and resource consumers have been proposed in [1, 31, 5, 15, 18]. The mechanism proposed in market based grid resource allocation include models such as: Auctions (based on outcome of a bidding process), Commodity markets (resource providers specify the charges and resource consumers pay according to the usage of resource), Tenders (based on contracting mechanism - that governs the agreement between resource providers and resource consumers), and Posted price (similar to commodity market except that there are special offers from time to time to attract more users) [5]. In addition to these economic models, different pricing schemes apply to the markets such as: Flat price model (mixed price for a period of time), Competitive economic models (dynamic pricing scheme), Usage timing (peak-o -peak - like telephone services), Prediction-based(based on the ability to predict responses by competitors), Loyalty-based (special prices for regular and loyal users), and Advanced contract based (contract established before resource use determines the price) [5]. Economic models have been used in the context of resource allocation, although the important question about which model is the most appropriate for supporting resource and allocation in distributed environment is difficult to pinpoint.

Commodity markets rely on polling aggregate supply and demand repeatedly to calculate the equilibrium price and all allocations are performed at this equilibrium price. As in Mini-Grids the resources are not dedicated and supply demand of resources is very dynamic, the complexity of implementing such centralized market which relies on the aggregate supply demand of resources becomes infeasible. Therefore, we have selected auction models as the platform for matchmaking of consumer and producer of resources in Mini-Grids. In the following section, we study different auction models.

3.4. Task Distribution Protocol

The Mini-Grid task distribution protocol is based on auctions. The Mini-Grid Framework uses a ‘resource-push’ approach of auction-based dynamic resource provision rather than the traditional ‘user-pull’ approach. In this ‘resource-push’ approach, the participating resource providers express their interest in executing a computational task dynamically by participating in the bidding process and thus eliminate the requirement of resource discovery mechanism.

The major steps in each phase of task distribution are detailed below. The task distribution involves 3+4+2 steps. The first three steps involve resource discovery, next four steps involve resource selection and task allocation, and the final two steps involve execution management as detailed below.

1. A BoT application generates tasks with a Task Context description, and submits them to the Resource Consumer.
2. The Resource Consumer announces each task to all Resource Providers currently listening to announcements using Messenger Component.
3. On receiving the announcement, the Resource Providers varies using its local Resource Context to see, if it can submit a bid. If not, it ignores the announcement.
4. The Resource Provider computes the bid based on the announced bidding strategy and its local Resource Context.
5. The Resource Provider submits the computed bid. Once the "Time-to-Bid" (TTB) prescribed time by which a bid must be submitted for consideration period elapses, the Resource Consumer proceeds.
6. The Resource Consumer evaluates the submitted bids and selects the optimal Resource Provider for execution of the task.
7. The Resource Consumer announces the winner. The winning Resource Provider gets the task from the TaskBus and executes it. The winning Resource Provider sends an acknowledgment to the winner notification.
8. On completion of the task execution, the Resource Provider sends a task completion notification, and returns the task including the result of the execution to the Task Bus.
9. Once the Resource Consumer gets the notification that the task has been executed, it collects the result from the Task Bus. The Resource Consumer sends an acknowledgment to Resource Provider for task completion notification. The submitter informs the application that task execution has been completed through Task Listener. The Task Listener retrieves the completed task from the submitter.

The pseudo-code for task allocation in the Resource Consumer is shown in algorithm 1. The algorithm proceeds if there is at least one Resource Provider interested in executing the task. If there is no Resource Provider, it times out and informs the application that it cannot schedule the task in Mini-Grid.

Algorithm 1: Task allocation on the Resource Consumer.

Input: Submitted Task.
Output: Winner notification

```
begin
forall the submittedTasks do
TaskContext taskContext = submittedTask.getTaskContext();
CALL auctioneer.createAuction(taskContext);
        time ← 0;
while time > timeToBid do
        submittedBids = CALL auctioneer.getSubmittedBids();
        winningBid = Min(submittedBids);
        end
        CALL messengerComponent.send(winnerNotication);
end
end
```

Algorithm 2: Algorithm for determining client participation in auction.

Input: Task submission notification
Input: Set of context statements detailing resource context
Output: Boolean value representing client in Resource Provider participation in bidding. TRUE when client can participate and FALSE otherwise.
begin

```
    forall the taskSubmissionNotication do
      TaskContext taskContext = notication.getTaskContext();
      ResourceContext resContext = ctxManager.getResourceContext();
      Avail ← Set of ContextStatement in ResourceContext;
      Req ← Set of ContextStatement in TaskContext;
      if ContextStatement   Req matches ContextStatement   Avail then
         return true;
      else
         return false;
      end
    end
end
```

The pseudo-code for determining client participation in Resource Provider is shown in algorithm 2. The client invokes the algorithm on receiving a task submission notification.

Since Mini-Grid is ad-hoc and extremely volatile, participating resources can leave at any time. In order to handle cases where an executing resource (a Resource Provider) fails or go offline, a simple failure handling mechanism has been implemented as part of the framework. Along with the task submission, the application specifies a value for Time-to-Live (TTL) parameter. TTL is a limit on the period of time. The resource consumer expects the resource provider to complete the execution of the task before this time elapses. If the resource provider does not send a task completion notification before the TTL elapses, the resource consumer assumes that resource provider has left Mini-Grid. Then the resource provider informs the application that it cannot schedule the task in Mini-Grid. The application can run the task locally on the resource consumer or reschedule the task in Mini-Grid.

3.5. Context-awareness for task and resource modelling

The coordinated resource sharing supplies a “context” that can be used to describe resources and requests. The context information, any information that can be used to characterize Grid entities, when modeled and managed introduces context-awareness into the Grid. This introduction of context-awareness into the Mini-Grid Framework enables provision of context-aware quality of service to the Mini-Grid applications. We have identified two main abstractions: tasks and resources and a range of user preferences

4. Evaluation

Each resource provider, participating in the auction process, need to decide the bid that it is going to submit. To define the bid, the resource provider needs to use the resource context. This involves context query and processing, which consumes CPU time. In our implementation context information are stored in a RDF triple store, a database for the storage and retrieval of subject-object predicate triples. The triple store needs to load data and respond to query over a knowledge base. Hence, time taken by the resource provider to define the bid depends on the query response time of the triple store. Query response time varies widely with different implementation. A review of the literature can provide many studies on performance of RDF triple store that are available. For example, Florian Stegmaier et al. [34] have stated that the execution time of query against database containing 100,000 triple set lasts for 28 milliseconds in Jena. However, other implementations like Oracle's Semantic Technologies consume even less time. This execution time would be 5-6% of the time-to-bid value. Hence, context processing time does not contribute much to the auction overhead. When the context processing time is higher than time to bid, the bidder (resource provider) would submit a delayed bid. In the meantime, once time-to-bid expires, the auctioneer (resource consumer) would ignore the delayed bid. Delayed bid can be identified by the combination of resource identifier and task identifier contained in the bid.

5. Conclusion

We used peer-to-peer techniques to implement a desktop grid system that eliminates the need for a centralized scheduling component. We have used a marked-based auction mechanism for dynamic task distribution. Traditionally in price-based mechanism, the price represents supply/demand condition of resources. However, we have used utility function based on application specific quality parameters for representing supply/demand condition. This approach enabled the use of context-based application specific quality parameters for dynamic task distribution. We have shown that a market-based auction mechanism can be used to compensate for absence the central server. Current implementation ensures that the context information supplied by default context monitors is consistent with the defined context model. The framework needs to be modified for ensuring consistency checks and conflict detection.

Acknowledgement

This research has been funded by the Danish Agency for Science, Technology, and Innovation under the project “PC Mini-Grids for Prediction of Viral RNA Structure and Evolution”, #09-061856.

References

1. D. Abramson, R. Buyya, and J. Giddy, A computational economy for grid computing andits implementation in the nimrod-g resource brok, CoRR, CS.DC/0111048 (2001).
2. K. Amin, G. von Laszewski, and A. R. Mikler, Toward an architecture for ad hoc grids, IEEE 12th Int. Conf. on Advanced Computing and Communications, ADCOM 2004, (2004).
3. Y. Amir, B. Awerbuch, and R. S. Borgstrom, The Java market: Transforming the Internet into a metacomputer, Technical Report CNDS-98-2, The Johns Hopkins University, 1998.
4. D. P. Anderson, BOINC: a system for public-resource computing and storage, in GC '04: Proceedings of the Fifth IEEE/ACM International Workshop on Grid Computing, New York, NY, USA, 2004, ACM Press, pp. 365-372.
5. R. Buyya, D. Abramson, J. Giddy, and H. Stockinger, Economic models for resource management and scheduling in grid computing, Concurrency and Computation: Practice and Experience, 14 (2002), pp. 1507-1542.
6. N. Cafferkey, P. D. Healy, D. A. Power, and J. P. Morrison, Job management in webcom, in ISPDC, IEEE Computer Society, 2007, pp. 25-30.
7. C. Catlett, P. Beckman, D. Skow, and I. Foster, Creating and operating national-scale cyberinfrastructure services, May 2006.
8. S. J. Chapin, D. Katramatos, J. F. Karpovich, and A. S. Grimshaw, The legion resource management system, in Proceedings of the Job Scheduling Strategies for Parallel Processing, IPPS/SPDP '99/JSSPP '99, London, UK, 1999, Springer-Verlag, pp. 162-178.
9. S. Choi and R. Buyya, Group-based adaptive result certification mechanism in desktop grids, Future Generation Comp. Syst., 26 (2010), pp. 776-786.
10. S. Choi, H. Kim, E. Byun, M. Baik, S. Kim, C. Park, and C. Hwang, Characterizing and classifying desktop grid, in Cluster Computing and the Grid, 2007. CCGRID 2007. Seventh IEEE International Symposium on, May 2007, pp. 743-748.
11. I. Foster, Globus toolkit version 4: Software for service-oriented systems, Journal of Computer Science and Technology, 21 (2006), pp. 513-520.
12. I. Foster and A. Iamnitchi, On death, taxes, and the convergence of peer-to-peer and grid computing, in In 2nd International Workshop on Peer-to-Peer Systems (IPTPS03, 2003, pp. 118-128.
13. I. Foster and C. Kesselman, Computational grids, (1999), pp. 15-51.
14. J. Frey, T. Tannenbaum, M. Livny, I. Foster, and S. Tuecke, Condor-g: A computation management agent for multi-institutional grids, Cluster Computing, 5 (2002), pp. 237-246.
15. J. Gomoluch and M. Schroeder, Market-based resource allocation for grid computing: A model and simulation, in Proceedings of the First International Workshop on Middleware for Grid Computing. Rio de, 2003, pp. 211-218.
16. M. Koh, J. Song, L. Peng, and S. See, Service Registry Discovery using GridSearch P2P Framework, Proceeding of CCGrid, 2 (2006), p. 1-1.
17. H. Kurdi, M. Li, and H. Al-Raweshidy, A classification of emerging and traditional grid systems, Distributed Systems Online, IEEE, 9 (2008), pp. 11.

18. K. Lai, L. Rasmusson, E. Adar, L. Zhang, and B. A. Huberman, Tycoon: An implementation of a distributed, market-based resource allocation system, Multiagent Grid Syst., 1(2005), pp. 169-182.
19. S. Lalis and A. Karipidis, Jaws: An open market-based framework for distributed computing over the internet, in Grid Computing GRID 2000, R. Buyya and M. Baker, eds., vol. 1971 of Lecture Notes in Computer Science, Springer Berlin / Heidelberg, 2000, pp. 87-106.
20. Laszewski, W. Smith, and S. Tuecke, A directory service for conguring high-performance distributed computations, in Proceedings of the 6th IEEE International Symposium on High Performance Distributed Computing, HPDC '97, Washington, DC, USA, 1997, IEEE Computer Society, pp. 365.
21. M. Litzkow, M. Livny, and M. Mutka, Condor - A Hunter of Idle Workstations, in Proceedings of the 8th International Conference of Distributed Computing Systems, IEEE Press, 1988, pp. 104-111.
22. L. Liu and N. Antonopoulos, From client-server to p2p networking, in Handbook of Peer-to-Peer Networking, X. Shen, H. Yu, J. Buford, and M. Akon, eds., Springer US, 2010, pp. 71-89.
23. T. Mattson, B. Sanders, and B. Massingill, Patterns for parallel programming, Software Patterns Series, Addison-Wesley Professional, first ed., 2004.
24. J. Maurer, A conversation with david anderson, Queue, 3 (2005), pp. 18-25.
25. N. Nisan, S. London, O. Regev, and N. Camiel, Globally distributed computation over the internet-the popcorn project, in Distributed Computing Systems, 1998. Proceedings. 18th International Conference on, May 1998, pp. 592-601.
26. T. V. Pham, L. M. Lau, and P. M. Dew, An adaptive approach to p2p resource discovery in distributed scientific research communities, Proceeding of CCGrid, 2 (2006), p. 12.
27. J. Sherwani, N. Ali, N. Lotia, Z. Hayat, and R. Buyya, Libra: a computational economy-based job scheduling system for clusters, Softw. Pract. Exper., 34 (2004), pp. 573-590.
28. D. Talia and P. Trunfio, A p2p grid services-based protocol: Design and evaluation, in Proceedings of Euro-Par 2004, M. Danelutto, D. Laforenza, and M. Vanneschi, eds., vol. 3149 of Lecture Notes in Computer Science, Springer Verlag, 2004, pp. 1022-1031.
29. P. Trunfio, D. Talia, H. Papadakis, P. Fragopoulou, M. Mordacchini, M. Pennanen, K. Popov, V. Vlassov, and S. Haridi, Peer-to-peer resource discovery in grids: Models and systems, Future Gener. Comput. Syst., 23 (2007), pp. 864-878.
30. C. A. Waldspurger, T. Hogg, B. A. Huberman, J. O. Kephart, and W. S. Stornetta, Spawn: A distributed computational economy, IEEE Trans. Softw. Eng., 18 (1992), pp. 103-117.
31. R. Wolski, J. S. Plank, J. Brevik, and T. Bryan, Analyzing market-based resource allocation strategies for the computational grid, International Journal of High Performance Computing Applications, 15 (2001), pp. 258-281.
32. J. E. Bardram and N. Venkataraman. The mini-grid framework: Application programming support for ad-hoc, peer-to-peer volunteer grids.
33. Neelanarayanan V. The Mini-Grid Framework: Application Programming Support for Ad hoc Volunteer Grids. 2013. 198 p. (ITU-DS; No. 97).
34. Florian Stegmaier, Udo Grobner, Mario Doller, Harald Kosch, and Gero Baese. Evaluation of current rdf database solutions. In Proceedings of 10th International Workshop of the Multimedia Metadata Community on Semantic Multimedia Database Technologies, SeMuDaTe09, 2009.
35. Abdullah, T., Sokolov, V., Pourebrahimi, B., Bertels, K. Self-organizing dynamic ad hoc grids. In Procedings 2nd IEEE International Conference on Self-Adaptive and Self-Organizing Systems Workshops, SASOW 2008 (2008).
36. Kerry Jean, Alex Galis, and Alvin Tan. Context-aware grid services: Issues and approaches. In International Conference on Computational Science (2004), pp. 166–173.