The Mini-Grid Framework: Application Programming Support for Ad-hoc, Peer-to-Peer Volunteer Grids

Jakob E. Bardram and Neelanarayanan Venkataraman

IT University of Copenhagen
Rued Langgaards Vej 7, DK-2300 Copenhagen S, Denmark
{bardram,nnve}@itu.dk

Abstract. Biological scientists work with bioinformatics algorithms that are either computational or data intensive in nature. Distributed platforms such as Grids and Peer-to-Peer (P2P) networks can be used for such algorithms. Classical Grid computing platforms are available only to a restricted group of biologist, since they are expensive, and require skilled professionals for deployment and maintenance. Due to its master-slave architecture, projects deployed using volunteer computing systems require 'slaves' to be convinced to participate. The alternative, P2P architecture is mainly used for data sharing. This paper presents the Mini-Grid Framework which is an P2P infrastructure and programming framework for distribution computational tasks like bioinformatics algorithm. The Mini-Grid Framework contributes with concepts and technologies for minimal configuration by non-technical end-users, a 'resource-push' auction approach for dynamic task distribution, and context modeling of tasks and resources in order to handle volatile execution environment. The efficiency of the infrastructure has been evaluated in several alternative experiments.

1 Introduction

The "Grid" refers to the vision of a hardware and software infrastructure providing dependable, consistent, fast, and inexpensive access to high-end computational capabilities [1]. Such a platform has great potential impact for many disciplines, such as bioinformatics. Generally speaking, grid computing platforms can be classified into two main categories; classic high-end grids and volunteer computing grids [2]. Classical grids provide access to large-scale, intra-and-inter institutional high capacity resources such as clusters or multiprocessors [1, 3]. However, installing, configuring and customizing such solutions require high technical knowledge and dedicated hardware and software resources. For these reasons, the deployment and operational cost of such systems are substantial, which prevents its adoption and direct use by non-technical users, such as biological researchers.

Volunteer computing systems, on the other hand, allow formation of parallel computing networks by enabling ordinary internet users to share their com-

puter's idle processing power [4, 5]. Such systems require setting up a centralized control system responsible for managing the contributed clients, who in turn periodically request work from a central server. Volunteer computing is highly asymmetric; it is a 'master-slave' architecture in which volunteers supply computing resource but do submit any work to be done. Public outreach and incentive structures (like high-score competitions) play a significant role in attracting volunteers.

The current approaches to grid computing is, as such, very centralized both technically and in use. Only a relatively few dedicated scientists use the classic grids like Globus and in volunteering grids, setting up projects is rather centralized and require significant technical skills and effort. In contrast, our goal is to create a distributed and ad-hoc approach for scientist to use parallel and distributed resources in their work. Our current work builds on creating support for bioinformatics analysis in biology laboratories. In this setting we would like to support biologists to utilize available computational resources on an ad-hoc basis. The main challenges in this setting is to support users with no technical knowledge to perform all the tasks involved in grid computing, i.e. locating available resources, distributing tasks and data, monitor progress, intervene if necessary, and to recollect the results for further use. And we want to do this on the available resource infrastructure, like the desktop and laptop PCs in the lab. Compared to existing approaches to grid technology, this kind of scenarios put up requirements for supporting:

- Ease of deployment and management of the infrastructure.
- Dynamic peer-to-peer resource discovery.
- Resource modeling which takes into account the context of the resources and the users.
- Resource models used for dynamic context-aware task distribution and scheduling.

This paper presents the Mini-Grid Framework, which is a runtime infrastructure and programming API enabling the creation of peer-to-peer and ad-hoc "mini-grids" in a local network environment. The main benefit of such a mini-grid infrastructure is that it is ready to use for the end-user without any configuration or management overhead; a user submitting a job simply exploits the devices visible nearby at the moment. This infrastructure is an important step in the direction of allowing non-technical scientists – like biologists – to interactively use a grid in their daily work.

2 Related Work

Large-scale grids often build on the Globus Toolkit [3], which is a middleware for handling distributed resources and delivering high-performance computational power to a restricted community of users with a specific goal. TeraGrid[1] is one

[1] www.teragrid.org

such example. Using Globus, NetSolve [6] and Ninf [7] allow the end-users to launch libraries installed on the remote machines from their applications using Grid enabled remote procedure calls. Based on a bag of service approach, Globus provides key services such as resource, data, and security management.

However, installing, configuring, and customizing Globus middleware requires a highly skilled support team, such as the London e-Science Centre[2] or the Enabling Grids for E-science project[3]. Participating in grid projects involves time consuming networking and training processes. Globus based grid computing infrastructure requires third party resource brokers or meta-schedulers for task distribution and follows a hierarchical client-server model for scaling.

Volunteer or Desktop Grids, in contrast, is designed to distribute computational tasks between desktop computers in e.g. a biology lab. Historically, the Condor project [4] pioneered using the idle time of organizational workstations to do parallel computing, but the best known infrastructure for volunteer grids is the Berkeley Open Infrastructure for Network Computing (BOINC) [5, 8]. BONIC is composed of a central scheduling server and a number of clients installed on the volunteers' machines. The client periodically contacts the server to report its availability and get workload. BOINC requires a fixed and static set of data servers deployed centrally that need to be maintained by each project team. Further, creating a volunteer computing application involves the process of obtaining and retaining volunteers, setting up high-profile initiatives like the the World Community Grid[4]. Other desktop grid technologies exists, like the Minimal Invasive Grid [9], and XtremWeb [10].

Condor [11], based on a master-slave architecture, has a centralized scheduling model and file-based configuration management (around 300 parameters). Condor discovers resources by advertisements [12] containing semi-structured data model – attribute/expression pairs which lack expressiveness – describing their capabilities.

All of these desktop grids are, however, using a client-server architecture consisting of a set of servers for project hosting, scheduling, and data management, and a set of clients which executes the workload. Commercial desktop grids, like Apple's Xgrid [13], uses similar centralized schedulers and controllers.

In order to address software architecture qualities like scalability, performance, and availability, recent research within software architecture have been addressing peer-to-peer (P2P) and hybrid software architectures. Such architectures help distribute resources while avoiding the vulnerability of a single-point-of failure inherent to the client-server architecture. From a utility point of view, the P2P approach helps utilize the resources (CPU, RAM, and disk) available of the distributed computers. From a deployment point of view, there is no need for installing and maintaining various centralized services.

Peer-to-Peer Grids distribute computation in a peer-to-peer fashion. For example, the OurGrid project uses a peer-to-peer topology between different labo-

[2] www.lesc.imperial.ac.uk
[3] www.eu-egee.org
[4] www.worldcommunitygrid.org

ratories [14]. Various protocols for supporting P2P service discovery (e.g. Gridnut [15] and GridSearch [16]) and P2P resource discovery [17] has been proposed. Organizing Condor pools in P2P network, requiring no central coordination, has been proposed in [18]. XtremWeb [19] envisages the use of peer-to-peer network for building volunteer computing platform. Our approach differs from existing P2P grids by addressing pervasiveness: mobility of resources, wireless connection between resources, and ability to adapt their surroundings – context awareness.

P-Grid [20] is a general purpose P2P distributed infrastructure which distribute algorithm by a rumor spreading technique for resource discovery and uses a pull-push model in its updating scheme. P-Grid also uses a reputation based security model and employs redundant peers for fault tolerance.

The Mini-Grid Framework falls within the P2P Grid category, but has a different focus; instead of supporting inter-lab P2P grids, we are targeting *intra-lab* P2P grids – i.e. supporting grids amongst desktop computers inside a lab. Moreover, we target a very volatile and ad-hoc nature of grid resources, including laptops which frequently enters and leave a network. The terms "ad hoc" and "volatile" refers to the nature of virtual organizations that can be formed by the mini-grid framework. We use a 'resource-push' approach for task distribution rather than complex resource discovery mechanism used in current P2P grids. Although we adopt a market-oriented resource management system in our auctioning system, this is merely for allocating tasks to different resource based on their capabilities. Overall, our approach is oriented towards 'social computing' that enables equal sharing resource and collaboration. This is in contrast to the 'traditional' approach to marked-oriented resource sharing in e.g. the Computational Clouds.

3 The Mini-Grid Framework

In contrast to desktop grids which runs on a fairly stable infrastructure (cabled networks, stationary desktop PCs, and dedicated servers), the Mini-Grid project is aiming at supporting the formation of *ad-hoc, peer-to-peer grids on an volatile infrastructure* based on whatever devices available, including portable devices on intermitted networking connections. The Mini-Grid thus supports the ad-hoc formation of grids based on e.g. students' laptop computers on a wireless LAN. But at the same time, the mini-grid could include more powerful nodes, like desktop PCs and dedicated servers, which would enable the mini-grid to scale to a more traditional grid using cabled and server-based computers. The Mini-Grid uses a 'resource-push' approach of auction-based dynamic task distribution rather than the traditional 'user-pull' approach. In this 'resource-push' approach (explained in section 3.2), the participating resources express their interest in executing a computational task dynamically and thus eliminates the requirement of resource discovery mechanism.

The Mini-Grid Framework is a runtime infrastructure and programming API for creating and running such mini-grids. This section provides an overview of Mini-Grid framework, including a description of its architecture, its auction

based task distribution and resource discovery, its context modeling of resources and tasks, and its extension points for specializing its behavior.

3.1 Architecture

A conceptual illustration of the Mini-Grid Framework is shown in figure 1. The main components are *Resource Providers* donating computational power, *Resource Consumers* using computational power, and the *TaskBus* responsible for distributing tasks and control information. A device participating in the mini-grid can be a Resource Provider, or a Resource Consumer, or both.

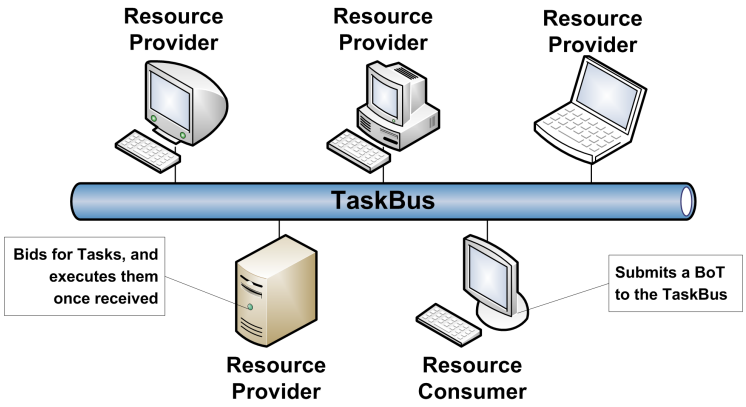

Fig. 1. A conceptual model of the Mini-Grid Architecture including the TaksBus and Resource Consumers and Resource Providers.

The framework is primarily designed for applications whose tasks are independent of each other, i.e. the so-called Bag-of-Tasks (BoT) applications, where each task is an atomic unit of execution. The business logic of a task is encapsulated in the execute method of the task.

As shown in figure 2, the Resource Consumer has three components; *Submitter*, *TaskBus*, and *Auctioneer*. Any BoT application interested in utilizing the infrastructure submits the tasks to the Submitter along with maximum life time of the task, called 'Time-To-Live' (TTL). TTL is a deadline fixed by the application within which it expects the results of the task. We assume that the application using the infrastructure does not know the execution time of a task in advance. The task include a *TaskContext* description, which includes the requirements for the target software and hardware. The TaskBus is used to announce new tasks in the grid, to collect bids in the auction process and to distribute tasks and results. Auction-based task distribution is delegated to an Auctioneer class that implements the auction strategy. The Auctioneer uses BidEvaluator to evaluated the submitted bids. The protocol of the Task Bus is detailed in section 3.2.

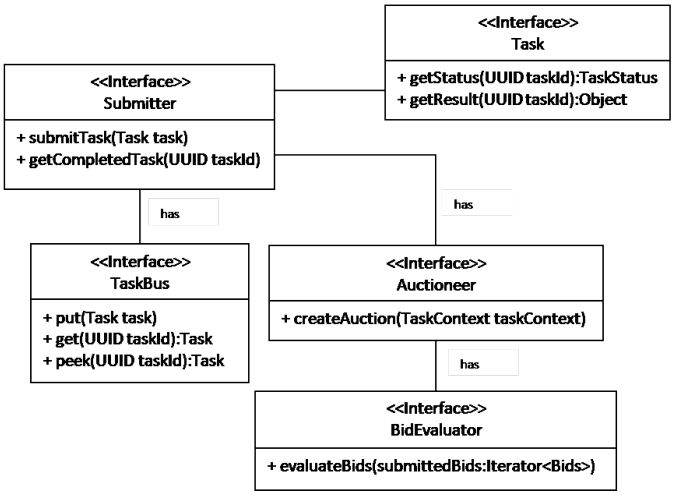

Fig. 2. Resource Consumer's Components

As shown in figure 3, the Resource Provider has five components; *Executor*, *TaskExecutor*, *Bidder*, *ResourceContext*, and the *TaskBus*. The Executor is responsible for managing the Resource Provider, including using the TaskBus to listen to task announcements, and using the Bidder to bid for tasks. The ResourceContext is responsible for checking if a task can be executed on this host and providing information for computing a Bid. The TaskExecutor is responsible for the execution of a task.

By extending the mini-grid framework, the application developer can specify different types of bids suitable for the particular application. Bids can, for example, be calculated based on the speed of a host, or its level of security. The resource provider computes the bid value by using the information provided by the ResourceContext.

Resource discovery is done as part of the auction process, and the mini-grid hence do not need a separate resource discovery mechanism. During the auction process, only resources that are currently participating in the mini-grid environment receive task requests and can submit bids.

3.2 Task Distribution Protocol

The Mini-Grid task distribution protocol is based on auctions. A constant named Time-to-Bid (TTB) specify the time that Resource Providers have to submit their bids. Task distribution involves 6+2 steps:

1. A BoT application generates tasks with a TaskContext description, and submits them to the Resource Consumer.
2. The Resource Consumer announces each task to all Resource Providers currently attached to the TaskBus.

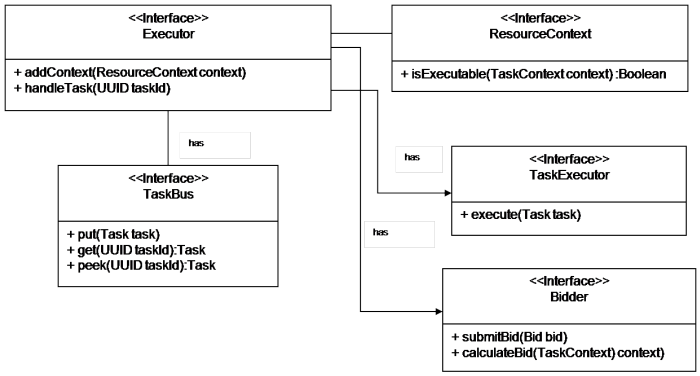

Fig. 3. Resource Provider's Components

3. On receiving the announcement, the Resource Providers check with its local ResourceContext to see, if it can submit a bid. If not, it ignores the announcement.
4. The Resource Provider submits a bid. Once the TTB period elapses, the Resource Consumer proceeds.
5. The Resource Consumer evaluates the submitted bids and selects the optimal Resource Provider for execution of the task.
6. The Resource Consumer announces the winner. The winning Resource Provider gets the task from the TaskBus and executes it.
7. On completion of the task execution, the Resource Provider sends a task completion notification, and return the task including the result of the execution to the TaskBus.
8. Once the Resource Consumer gets the notification that the task has been executed, it collects the result from the TaskBus.

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 the Mini-Grid environment.

Since the mini-grid environment is ad-hoc and extremely volatile, participating resources can leave at any time. In order to handle cases where an executing node (a Resource Provider) fails or go offline, a simple failure handling mechanism has been implemented as part of the mini-grid framework. Along with the task submission, the application specifies a wall-time, an approximate estimation of run time of the task. The resource consumer expects the resource provider to complete the execution of the task before this time elapses. If the resource provider has not send a task completion notification before the wall-time elapses, the resource consumer assumes that resource provider has left the mini-grid environment. Then the resource provider re-auctions only the task for

Algorithm 1 Task allocation on the Resource Consumer

Require: Task submitted to distributed submitter
Require: At least one Provider is interested in executing the task
1: **for all** $submittedTask$ **do**
2: CALL auctioneer.createAuction($taskContext$)
3: $time = 0$
4: **while** $time > timeToBid$ **do**
5: $submittedBids$ = CALL $auctioneer.getSubmittedBids()$
6: $winningBid = Min(submittedBids)$
7: **end while**
8: CALL auctioneer.notify($winningBid$)
9: **end for**

which it has not received the task completion notification. The application can specify the number of retires the resource provider can perform. In case the resource provider fails to receive task completion notification in all retries, then the application is notified. Delayed task completion notifications are ignored by the resource consumer.

3.3 Context Modeling

The context model in the Mini-Grid Framework is a formal model of the characteristics of an entity. Context is based on the notion that entities have properties, and properties have values. Entities can be described by making statements that specify the properties of the entity and their values. For example, "Computer A has Intel Core 2 Extreme QX9650 processor" is a statement used to define the hardware context of Computer A. Such subject-predicate-object expressions describes the properties of the entity.

This approach provides expressiveness over the attribute/expression pairs used in Condor's ClassAds. This expressiveness permits semantic matching using domain knowledge expressed as concepts. For example, using attribute/expression mechanism a resource's operating system can be defined like "OS = value". When an application requires devices having operating system compatible to Unix, it has to issue a query requesting for disjunction of all Unix compatible operating system such as;

```
OS=Linux || OS=Solaris || OS=IRIX ||...
```

Defining such disjunctive set for abstract concepts may contain a number of elements making the exact matching process, used in traditional resource matching, more constrained. Instead, in our approach, Unix can be defined as a subclass of operating system and all variants of Unix operating systems can be defined as a type of Unix operating system. Using RDF, the above information can be represented as; "Unix is a operating system"; "Solaris is a type of Unix operating system"; etc. Then compatibility concept in domain knowledge can be defined as rules;

```
[compatibleOperatingSystem: ?p rdf:type ?q, ?q rdfs:subClassOf
operatingSystem -> ?p compatibleOS ?q]
```

Thus, the Mini-Grid Framework enables participation of resource providers in an auction based on semantic reasoning rather than simple attribute matching. This approach is very similar to the work done by John Broke et al. [21]. However, their approach is centralized whereas our approach is distributed.

In the Mini-Grid framework, resources and tasks have context. Currently, we are using context modeling strictly for technical information about required hardware and software. However, the model is sufficiently generic to include other types of context information in the bidding and execution protocol, including information of the usage context of the device. For example, it can model context like location of the device and what it is being used for. This will allow the bidding process to avoid scheduling 'important' tasks on devices like student's laptops which may be removed from the laboratory.

3.4 Framework Extensions

The Mini-Grid Framework is highly extensible in central places. For example, the default implementation of the BidEvaluator, schedules a task on the fastest resource possible. But by extending or overwriting this implementation, the application developer can define other goals such as scheduling the task on resource that has better networking capabilities, has a more stable configuration, or is trusted.

Similarly, the default auction strategy is a 'First Price Sealed Bid Auction' [22] in which bidders are not aware of each others' bid value and runs only a single round. When the bidders receive a bid request, they can determine the bid value based on their capability. First-price sealed bid auctions has minimal communication overhead. However other types of auction can be used by extending the framework.

4 Implementation

The Mini-Grid Framework has been implemented in Java. The implementation of the TaskBus uses UDP multicasting for exchanging meta-information about tasks and bids among the participating devices, and uses TCP connections for exchanging tasks between submitters and executors. The TaskExecutor is based on a thread pool, and a simple sequential single-item auction (i.e., tasks are auctioned one at a time) has been implemented. Though bidding and clearing the auction is simple, it can miss the optimal allocation. However, combinatorial auctions where all tasks are auctioned and the bidding happens based on groups of tasks, can be implemented, but with increased complexity and communication overhead. By default the framework supports computation of the bid based on the current load of the resource. However, the framework supports definition of new types of bids suitable for the application being developed by extending the

Bid interface. The current implementation of the auction-based task distribution protocol is sequential; i.e., tasks are auctioned in their submission order. However, the auctioneer and the application are tied up only for the duration of the task announcement. Bid submission and evaluation are asynchronous. Thus the frameworks permits concurrent task submission with minimal waiting time.

Deployment of the mini-grid environment requires minimal configuration compared to existing approaches. The framework requires a multicast address for UDP communication and port numbers for TCP/UDP communication at the time of deployment. Further the framework requires proper configuration for enabling TCP/UDP communication at firewalls and other network equipments. These configuration requirements are similar to what is required for users to share their music collection via Apple iTunes.

The prototype has been integrated with the CLC Bio Workbench for bioinformatics research[5]. This implies that users of the CLC Bio workbench has the option of executing their bioinformatic algorithms (e.g., BLAST or Tree Alignment) in a Mini-Grid environment. A pilot deployment for evaluating the framework is underway at the Danish iNano research centre[6].

5 Evaluation

In order to asses the effectiveness of the Mini-Grid framework, we performed an empirical evaluation in which we compared its performance with a single machine. The performance is measured on two parameters:

Average waiting time – the average time elapsing from the arrival of the task at the Submitter to it starts execute at the Executor.

Average turn-around time – the average time elapsing from the arrival of the task at the Submitter to its completion at the Executor.

For our study, the Mini-Grid ran on a set of 21 identical Intel Core Duo 2.33GHz desktop PCs with 2GB RAM, running MS Windows in a 100Mbs LAN. Each node could play both the role of resource provider and resource consumer, but for simplification one node was configured to act as a resource consumer, and the rest as resource providers with varying resource context. For each auction, the resource providers submitted different bids by using a random number generator. Tasks arrived at the Submitter in batch. For the experiment we have considered computational intensive tasks, i.e., the tasks keep the cpu busy for the period of their execution time and sequential single task auction for task distribution.

The main overhead in Mini-Grid approach is the time taken for auctioning the tasks. Hence in order to study the overhead in the mini-grid approach, we conducted three experiments that shows the impact of TTB on the average waiting time of a task. In these experiments, we varied the TTB and measured the average waiting time for a fixed number of tasks and nodes. Then we repeated

[5] www.clcbio.com
[6] www.inano.dk

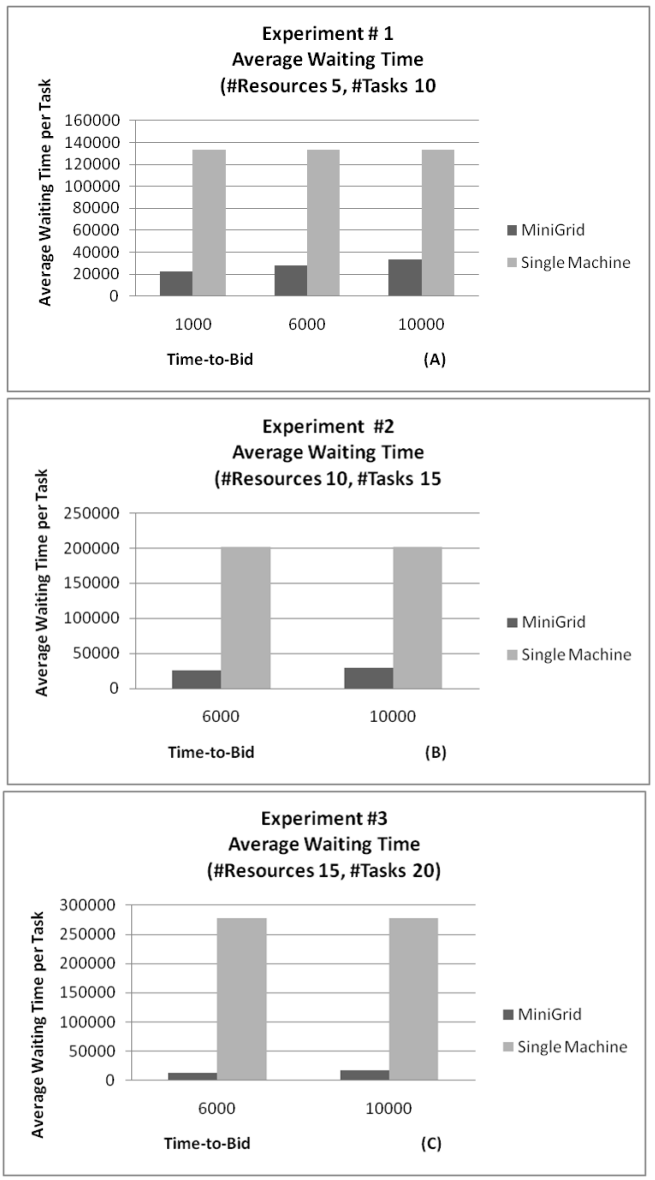

Fig. 4. Experiment #1– #3: Average Waiting Time per Task

the experiment for another fixed number of tasks and resources. The results of these experiments are shown in figure 5.

In experiment #1, 5 nodes participated and 10 tasks were submitted, and the TTB was 1,000, 6,000, and 10,000 milliseconds respectively. The experiments were repeated several times and average numbers are presented here for discus-

sion. The results shows that the TTB influence the average waiting time by a proportional constant amount of time. The result implies that the Mini-Grid application developer has to choose the TTB based on the type of network used for deployment. For example, in a wired local area network, a TTB value of 200 ms would be sufficient. Here we have considered different values for TTB, in order to study the behavior in different types of network.

Experiment #2 and #3 were conducted to compare the performance of the Mini-Grid Framework with single Intel Core Duo 2.33GHz desktop PCs having 2GB RAM. The second experiment was conducted with TTB = 6000 ms, using 10 nodes and 15 tasks, and 15 resources and 20 tasks, measuring the average waiting time for each task. We have computed the average waiting time for the same set of tasks when there is only one Executor, i.e. the case of a single machine. However, the single machine approach do not have the overhead of TTB. The result show that the mini-grid infrastructure outperforms a single machine.

Experiment #4 measured the average completion time by having fixed value for TTB (10000 ms) and task execution time (1000 sec.). We varied the number of resources participating in the Mini-Grid and the total number of tasks submitted. The results are shown in figure 5. From the graph, we find that the average completion time changes only when the number of tasks is a multiple of number of resource providers. The average completion time decreases with increase in participation of resource providers.

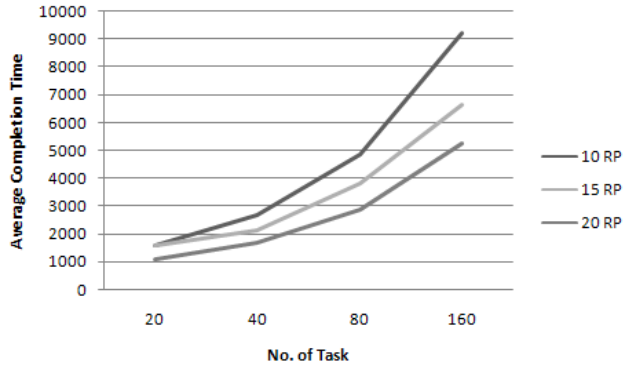

Fig. 5. Experiment #4: Average Completion Time per Task

When a set of task is concurrently submitted, then the auction mechanism increase waiting time. Experiment #5 was conducted to measure this waiting time. The x-axis provides number of tasks submitted concurrently and the y-axis represents the average waiting time of each task in the concurrent submission. The results are shown in figure 5. The waiting time is directly proportional to number tasks submitted concurrently. When the execution time of individual

tasks are longer and small number of task are submitted concurrently, then this waiting time is insignificant. When large number of tasks with short execution time are submitted concurrently, this waiting time will affect the completion time of the task. However, the framework permits the application developer to apply other, more suitable auctioneer, if needed.

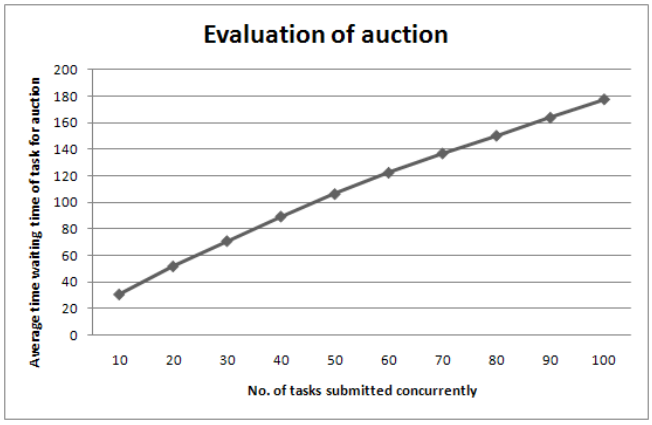

Fig. 6. Experiment #5: Impact of concurrent task submission during auctioning

6 Conclusion

In this paper we have presented the design, implementation and evaluation of the Mini-Grid Framework which includes an auction based scheduling algorithm for bag-of-tasks applications on ad-hoc Grids. In comparison to previous approaches published in the literature, the Mini-Grid Framework eases deployment and management of the infrastructure, dynamically discovers resources using peer-to-peer strategy, uses an auction-based task distribution protocol, and models execution context in a way which allow for semantic reasoning in the scheduling process. The framework is highly extendable and can be adapted to several types of applications.

The framework was evaluated by comparing the performance of our framework's auction based task allocation to the task allocation in the single machine. The evaluation showed that a mini-grid will exploit available resources for parallel computing, thereby out-performing a single machine. The main overhead is associated with the time used for the bidding process in the task auction.

As an extensible framework, the Mini-Grid Framework has the fundamental building blocks for creating application that distribute bag-of-tasks in a medium size ad-hoc network of volunteer computers. Ongoing work is concerned with the design of context-aware scheduling and contingency management. Context-aware

scheduling will enable scheduling that takes into account the physical setting of the involved nodes. Contingency management will help us run on more volatile and heterogeneous infrastructures. For example, allowing asynchronously task and result distribution which would allow the resources to be off line for a period of time. Currently bioinformatics algorithms (including e.g. BLAST) are being implemented as a part of the CLC Workbench to use the Mini-Grid infrastructure. This will enable us to leverage bioinformatics analysis in a biology lab using state-of-art user-friendly bioinformatic software and also to explore hardware accelerated bioinformatics algorithms running on the CLC Cube[7] and the CLC Cell[8]. Peer-to-peer cooperation across LANs has not yet been implemented, but will be addressed using inter-LAN gateways.

References

[1] Foster, I., Kesselman, C.: Computational grids. (1999) 15–51
[2] Kurdi, H., Li, M., Al-Raweshidy, H.: A classification of emerging and traditional grid systems. Distributed Systems Online, IEEE **9**(3) (March 2008) 1–1
[3] Foster, I.: Globus Toolkit version 4: Software for Service-Oriented Systems. Journal of Computer Science and Technology **21**(4) (2006) 513–520
[4] Litzkow, M., Livny, M., Mutka, M.: Condor - A Hunter of Idle Workstations. In: Proceedings of the 8th International Conference of Distributed Computing Systems, IEEE Press (1988) 104–111
[5] Anderson, D.P.: 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, ACM Press (2004) 365–372
[6] Seymour, K., YarKhan, A., Agrawal, S., Dongarra, J. In: NetSolve: Grid Enabling Scientific Computing Environments. Volume 14 of Advances in Parallel Computing. Elsevier (2005) 33–51
[7] Y, Y.T., Nakada, H., Sekiguchi, S., Suzumura, T., Matsuoka, S.: Ninf-g: A reference implementation of rpc-based programming middleware for grid computing. Journal of Grid Computing **1** (2003) 41–51(11)
[8] Anderson, D.P., Christensen, C., Allen, B.: Grid resource management—designing a runtime system for volunteer computing. In: SC '06: Proceedings of the 2006 ACM/IEEE conference on Supercomputing, New York, NY, USA, ACM Press (2006) 126
[9] Vinter, B.: The Architecture of the Minimum intrusion Grid, MiG. In Broenink, J., Roebbers, H., Sunter, J., Welch, P., Wood, D., eds.: Communicating Process Architectures. IOS Press (2005)
[10] Fedak, G., Germain, C., Neri, V., F.Cappello: XtremWeb: a generic global computing system. In: Proceedings of First IEEE/ACM International Symposium on Cluster Computing and the Grid, IEEE Press (2001) 582–587
[11] Thain, D., Tannenbaum, T., Livny, M.: Distributed computing in practice: the condor experience. Concurrency - Practice and Experience **17**(2-4) (2005) 323–356
[12] Raman, R., Livny, M., Solomon, M.: Matchmaking: distributed resource management for high throughput computing. High Performance Distributed Computing, 1998. Proceedings. The Seventh International Symposium on (Jul 1998) 140–146

[7] www.clccube.com
[8] www.clccell.com

[13] Kramer, D., MacInnis, M.: Utilization of a local grid of mac os based computers using xgrid. In: 13th IEEE International Symbosium on High Performance Distributed Computing. (2004) 264–275
[14] Cirne, W., Brasileiro, F., Andrade, N., Costa, L., Andrade, A., Novaes, R., Mowbray, M.: Labs of the World, Unite!!! Journal of Grid Computing **4**(3) (2006) 225–246
[15] Talia, D., Trunfio, P.: A P2P Grid Services-Based Protocol: Design and Evaluation. In Danelutto, M., Laforenza, D., Vanneschi, M., eds.: Proceedings of Euro-Par 2004. Volume 3149 of Lecture Notes in Computer Science., Springer Verlag (2004) 1022–1031
[16] Koh, M., Song, J., Peng, L., See, S.: Service Registry Discovery using GridSearch P2P Framework. Proceeding of CCGrid **2** (2006) 11
[17] Pham, T.V., Lau, L.M., Dew, P.M.: An Adaptive Approach to P2P Resource Discovery in Distributed Scientific Research Communities. Proceeding of CCGrid **2** (2006) 12
[18] Butt, A.R., Zhang, R., Hu, Y.C.: A self-organizing flock of condors. J. Parallel Distrib. Comput. **66**(1) (2006) 145–161
[19] Fedak, G., Germain, C., Neri, V.: Xtremweb: A generic global computing system. In: In Proceedings of the IEEE International Symposium on Cluster Computing and the Grid (CCGRID01, Press (2001) 582–587
[20] Aberer, K., Cudré-Mauroux, P., Datta, A., Despotovic, Z., Hauswirth, M., Punceva, M., Schmidt, R., Wu, J.: Advanced peer-to-peer networking: The P-Grid System and its Applications. PIK Journal - Praxis der Informationsverarbeitung und Kommunikation, Special Issue on P2P Systems (2003)
[21] Brooke, J., Fellows, D., Garwood, K., Goble, C.: Semantic matching of grid resource descriptions. In: In Proceedings of the European Across Grids Conference, 2004, http://www.Grid-interoperability.org/semres.pdf, Springer (2004) 240–249
[22] Klemperer, P.: 1. In: Auctions: Theory and Practice. Princeton University Press (2004)