Self-Organizing Dynamic Ad Hoc Grids[*]

Tariq Abdullah, Vassiliy Sokolov, Behnaz Pourebrahimi, Koen Bertels
Computer Engineering, EEMCS, Mekelweg 4, 2628CD, Delft, The Netherlands
{m.t.abdullah, b.pourebrahimi, k.l.m.bertels}@tudelft.nl, v.sokolov@student.tudelft.nl

Abstract

Resource management for ad hoc grids is challenging due to the participation of heterogeneous, dynamic, autonomous and ephemeral nodes. Different underlying network infrastructures, and varying use and access policies make resource management even more complex. Therefore it is required to develop such a resource management mechanism that will enable the ad hoc grid to self-organize according to the workload of the resource manager. The proposed mechanism is based on the emergent behavior of the participating nodes and adapts with respect to changes in the ad hoc grid environment. Scalability and robustness of the proposed mechanism is tested by running the experiments on PlanetLab. Simulation results show that our mechanism performs better than previously proposed mechanisms.

1. Introduction

Ad hoc grids have geographically distributed, heterogeneous, and ephemeral nodes. The participating nodes may have different ownership with varying use-policies. Resource management for nodes with above characteristics becomes a challenging undertaking. Different underlying network infrastructure and variable participation of nodes further increases the administrative complexity of the ad hoc grid, and hence resource management in ad hoc grid becomes even more complex. Therefore it is required to develop such a resource management system that will enable the ad hoc grid to manage and self-organize itself under varying workloads. The context of the research presented in this paper aims to understand and develop appropriate mechanisms for self management and organisation. Where P2P and fully centralised systems are often considered to be mutually exclusive and residing on both ends of an infrastructural spectrum, we consider them to be part of a continuum where the system should be capable of restructuring itself in either of these states, or any intermediate state between those two extremes. To this purpose, we inspire ourselves on micro-economic, market based mechanisms where all the information available at individual node level condenses into a simple global metric, namely the price. In economics in general and in financial markets in particular, it is an accepted axiom that the market price incorporates all available information, thus reflecting the overall state of the system. The way the basic market mechanism operated, is very simple: the price of a good goes up when there is more demand than supply and it goes down when there is more supply than demand. By analogy, we define the price of a resource to reflect the scarcity (abundance) of that particular resource in the system. But not only the price of a good is computed, the cost related to find the appropriate good should also be taken into account. The transaction cost attached to submitting a request/offer by a node to the matchmaker will also be computed.

In this paper, we use one of the above mentioned micro-economic approach to dynamically add and remove matchmakers. We envision that the ad-hoc grid could have one or more matchmakers according to the size of the grid and consequently, the number of requests sent by the individual nodes. For instance, as the grid grows, the number of nodes sending requests may become too big for one matchmaker to handle, thus requiring a second (or more) matchmaker to become active to assist the primary matchmaker in its task. As the grid shrinks, the system should be able to demote matchmakers and return to a state with less matchmakers. This mechanism allows the grid to modify its infrastructure independent of the context in which it operates. The question then boils down to defining these upper and lower bounds that will incite the system to, respectively, add or remove matchmakers to (from) the system. If the cost for making a transaction becomes excessively high, because of the incapability of the matchmaker to process all requests, then the grid itself becomes completely ineffective as it is no longer capable of using the available resources for the tasks to be executed by the grid. Where the use of markets and auctions is by no means original, its use in the context of such self organisation constitutes its main innovation. We

[*]The authors are thankful Jeff Hoye, from Max Planck Institute for Software Systems, for his help about freepastry API

will show that the simple basic mechanisms as described above, allows the system to automatically scale its infrastructure to be better suited for its current,and dynamically changing state.

To this purpose, we compute the cost of a single request made to the matchmaker, referred to as the Transaction Cost(TCost), which is calculated for every request/offer message being submitted to the matchmaker (Refer Section-3 for details of TCost). In this paper, we describe experiments that define and calculate the upper and lower threshold of the matchmaker workload (TCost) in the ad hoc grid. The calculated upper and lower threshold values are used to dynamically increase or decrease the number of matchmakers. The results reported in this paper are obtained from experiments run in PlanetLab. These results show that the mechanism enables the ad hoc grid to self-organize according to the workload conditions in the ad hoc grid. The following simplifying assumptions are adopted in this paper and which will be relaxed in future work:

- We only consider the state with one or more matchmakers and do not look at P2P issues; Consequently, we do not consider the case where the single matchmaker breaks down and the system goes into a P2P state;

- We do not address the issue of routing of requests when more (or less) matchmakers are introduced nor address the issue of how to create (or delete) new grid segments. We assume the necessary extensions to an overlay network that address this problem are available.

- We assume that there is a known subset of nodes in the grid that are potential candidates for being promoted to matchmaker;

The rest of the paper is organized as follows. Section-2 provides an overview of the related work. Section-3 describes the proposed mechanism in detail. Section-4 explains the experimental setup and discusses the results. Section-6 concludes the paper and discusses future work.

2. Related Work

Choi et al.[4] proposed an adaptive model for performing group-based scheduling in an ad hoc grid and called those groups as *volunteer groups*. The volunteer groups are defined according to the individual volunteer properties. Each volunteer group coordinates with the volunteer server, via the volunteer group coordinator, to perform scheduling. The volunteer server is a single point of failure in this approach and may become a performance bottleneck. Minimum-delay Dynamic Tree (MDTree)[12] organized nodes in the form of a tree, based on the link delay of each joining node. MDTree split a grouped set of nodes into subgroups when number of nodes exceeds a certain number of nodes in a group. Fixed sized groups and the use of link delay as group formation parameter are the drawbacks of this approach. Zhou et al.[13] discovered idle processing cycles and grouped them into geographic and night time aware overlay networks. Unfinished tasks are migrated to another night time zone overlay network. The main drawback of this work is that the host availability model is not based on the resource requirements of the job. Furthermore job migration may result in communication overhead.

Attribute encoding for resource discovery in DHT based overlay networks is studied in[6]. The majority of the encoded attributes may be mapped to a small set of nodes in the overlay network, therefore attribute encoding may result in a load imbalance condition. Erciyes et al.[5] propose a dynamic load balancing middleware protocol for the gird, in which each cluster coordinator first attempts to balance the load in its own cluster and when it fails, then it coordinates with other cluster coordinators to transfer/receive the workload. Padmanabhan et al.[2] proposed a self-organized grouping method that forms and maintains autonomous *resource groups*, according to some pre-specified resource characteristics, where each group contains a set of similar resources. The main drawback of their work is that there is no load balancing mechanism among the groups formed. Butt et al.[3] implemented a P2P based Condor flocking to share resources in different Condor pools[11]. They did not consider the dynamic introduction/removal of Condor pools or the workload condition of a Condor pool manager. Peermart[7] used one matchmaker for each type of resource being traded in the ad hoc grid. A new matchmaker is introduced only when a new resource type is introduced. Peermart did not consider the overload condition of a matchmaker for the introduction of new matchmakers.

The above discussed approaches use different parameters to distribute the workload of one matchmaker among multiple matchmakers. These parameters are volunteer properties[4, 2], attribute encoding[6], a matchmaker for each resource type[7], or attempt to find the best matchmaker[3] from a fixed pool of matchmakers. As these approaches do not consider the workload of the matchmaker(s), they may end up with an inappropriate infrastructure for the given state of the grid. Although the work presented in[12, 5] attempt to form fixed sized node groups or attempt to balance the workload of fixed number of cluster, these approaches do not discuss the infrastructure self-organization in ad hoc grid.

3. Proposed mechanism

Starting point of our model is the use of a Continuous Double Auction (CDA) as the matchmaking mechanism[8].

Attached to each request is a transaction cost (TCost) which reflects the workload of the matchmaker. The node submitting a request is supposed to pay this TCost to the matchmaker. To this purpose, every node is given an initial budget which can be used to this purpose.

The promotion of a node to a matchmaker is based on the workload of the current matchmaker(s). The matchmaker workload can be calculated in different ways. Even though every choice of cost function has some arbitrary aspect to it, we are looking for a simple and efficient one. We simply count the number of messages to be processed by the matchmaker before processing the newly received request/offer message[1]. This TCost is calculated for each request/offer. The Matchmaker agent maintains request and offer repositories in respectively descending and ascending order of the bid and ask prices. A new request/offer message is placed in request/offer repository at its proper place by using insertion sort. If there are N requests/offers and the newly received request/offer is placed at index L , where $0 < L < N$, then

$$requests = R_1 > R_2 > R_3 > ...R_{L-1} > R_L > R_{L+1} ... > R_N > R_{N+1}$$

$$offers = O_1 < O_2 < O_3 < ...O_{L-1} < O_L < O_{L+1} ... < O_N < O_{N+1}$$

TCost for a request/offer message:

$$TCost_{Request} = Count(R_1...R_{L-1})$$

$$TCost_{Offer} = Count(O_1...O_{L-1})$$

Where $Count()$ is the number of the entries in the list. The TCost value for a matched request/offer pair is the average of the their individual TCost values.TCost for matched pair is calculated as:

$$MatchedPairTCost = (TCost_{Request} + TCost_{Offer})/2$$

The matchmaker periodically calculates the average TCost which is calculated as follows:

$$AvgTCost = (\sum_{i=t_0}^{t} TCost(i)/(\sum_{i=t_0}^{t} N(i)$$

Where $\sum_{i=t_0}^{t} TCost(i)$ is the sum of TCost of the messages processed in time interval $[t_0, t]$ *and* $\sum_{i=t_0}^{t} N(i)$ is the total number of messages processed in this interval. A matchmaker promotes a node as a matchmaker or demotes a matchmaker back to normal node when its average TCost is above or below the matchmaker's *upper threshold* (see Section-4). In principle, the transaction cost could become infinitely high which would inhibit the grid to be operational as no matching could occur anymore. It is therefore important to have mechanisms to counter such an event. This is exactly the goal of the experiments described below.

[1]also used by[5] to calculate the workload.

3.1. System Architecture

Our ad hoc grid consists of autonomous, dynamic, volatile and loosely connected nodes that can join, leave or change their roles whenever needed. Each node is composed of three agents: *Consumer*, *Producer* and *Matchmaker*. Communication between different nodes is done through the underlying structured overlay network via *communication module*. The specification of these agents is summarized below:

- Consumer Agent: The consumer agent estimates the task execution time, the required resource quantity, and the bid price in its *task manager* module. The consumer agent submits the job to the producer agent and receives the job results in its *job Manager* module. As stated above, each consumer is given an initial budget it can use for buying resources and to pay the TCost for each request submitted.

- Producer Agent: The producer agent estimates about the idle available resource quantity, and the ask price in its *resource manager* module. Receiving of jobs from consumer agents, execution of consumer jobs and returning of results is done in *job Manager* module. Similar to the consumer, every producer also has an initial budget to which is added the renumeration it receives when its resources are actually spent and from which is subtracted the TCost it has to pay when submitting a request to the matchmaker.

- Matchmaker Agent: The matchmaker receives the requests/offers from the consumer/producer agents and inserts the received request/offer in its request/offer buffers as explained above. This task is performed by the *repository manager* module. The matchmaking process is performed in the *matchmake* module.

- Segmenter: This module is part of the matchmaker. It contains the logic for calculating the TCost. It also makes the decision to promote a node as matchmaker and vice versa according to the workload (TCost) of the matchmaker. It is also responsible for sharing the workload of one matchmaker with other matchmaker(s), which is above the TCost *upper threshold* value of one matchmaker. The segmenter module does so by forwarding the request/offer message to the other matchmaker(s). This module also manages the communication between different matchmakers.

Complete details of the system agents can be found in our previous work[8]. The pricing function for calculating bid/ask price is discussed in Section-3.3

3.2. Continuous Double Auction (CDA)

Double auctions are one of the many-to-many types of auctions. CDA supports simultaneous participation of producer/consumer, observes resource/request deadlines and can accommodate the variations in resource availability. Several agents (say consumers) can initiate an auction and several other agents (say producer) can bid in the auction. The resource requests are called the bids and the resource offers are called the asks. Auctioneer (matchmaker) collects bids/asks and matches them immediately on the detection of compatible bids. A compatible bid is a pair of resource offer and resource request where task constraints (such as resource quantity, time deadline, price) are satisfied. The auctioneer finds the matches between the buyers and sellers by matching offers (starting with lowest price and moving up) with requests (starting with highest price and moving down). Consumers/producers do not have global information about the supply and demand and buyers and sellers are not aware of the other's bids or asks and. They submit their asks/bids based on their local knowledge. When a task query is received by the matchmaker, the matchmaker searches all available resource offers and returns the best match. If no match is found, the bid/ask is stored in matchmaker repositories till the time to live (TTL) for them is expired or a match is found.

3.3. History-based Dynamic Pricing

In the proposed set of experiments, consumer and producer agents join the ad hoc grid with an initial pre-defined price and dynamically update it over the time using a history-based dynamic pricing strategy. The proposed price is a reflection of the value of each resource unit which the consumer and producer agents are willing to buy or sell. The pricing function is explained in previous work[9]. An overview of the pricing function is given here. The agents perceive the demand and supply of the resources through their previous experiences and update their prices accordingly. Based on this strategy, ask and bid price at time interval t are defined as:

$$P(t) = P(t-1) + \triangle P$$

Where $P(t)$ is the new price and $P(t-1)$ denotes the previous price. $\triangle P$ for seller and buyer is defined according to the previous resource/task utilization history.

$$\triangle P \text{ for seller: } \triangle P = \alpha(\mu(t) - \mu_{thR})p(t-1)$$
$$\triangle P \text{ for buyer: } \triangle P = \beta(\mu_{thT} - \mu(t))p(t-1)$$

Where α and β are the coefficients to control the rate of price change. In this paper $\alpha = \beta = 0.8$ is used. μ_{thT} and μ_{thR} are task and resource utilization threshold. $\mu_t$is task/resource utilization of the individual node. μ_t is defined as:

$$\mu_t = \sum_{i=t_0}^{t} x(i) / \sum_{i=t_0}^{t} N(i)$$

Where $\sum_{i=t_0}^{t} x(i)$ is the sum of sold/purchased resources in time period$[t_o, t]$ and $\sum_{i=t_0}^{t} N(i)$ is the total number of offered/requested resources in time period $[t_o, t]$. The transaction price is then computed as the average of the bid and ask prices. Note that the transaction price is the amount that a consumer will pay to the producer for consuming the producer resources and the Transaction cost(TCost) represents the matchmaker workload.

4. Experimental Setup

The proposed mechanism is implemented on top of Pastry[10], a structured P2P overlay network, and was tested on PlanetLab[1]. Pastry is used for node joining/leaving, for lookup of matchmaker(s), for node-to-node and node-to-matchmaker communication. We refer to [10] for Pastry details.

We used PlanetLab[1] to run our experiments. PlanetLab is an open, geographically distributed computing environment/test-bed. Each project runs in its own network of virtual machines, called *slice*. A slice isolates projects from each other. Moreover there is no centralized control over resources in PlanetLab. We looked at the situation when the network is low on work and high on resources, referred to as Resource Intensive Network (abbreviated as RIN, where tasks $<<<$ resources). The task-resource ratio was 20%-80% in RIN.

The matchmaker's throughput is determined in terms of its matchmaking efficiency, transaction cost (TCost) and the response time. Matchmaking efficiency is calculated in terms of request/offer utilization (calculated as $\sum matchedRequest$ / $\sum request * 100$ and $\sum matchedOffer$ / $\sum offer * 100$ respectively). The transaction cost (TCost) calculation is described in Section-3. The matchmaker response time is the time interval between receiving a request/offer message by the matchmaker and the time instance when matchmaker has processed a request/offer message. The number of nodes was varied from 15-650 and the number of matchmakers was varied from 1-4 in the experiments. The workload was managed in such a way that the maximum number of matchmakers were needed and then gradually decreased to provoke the demotion of matchmakers to normal nodes again. The Job execution time, job deadline and required/offered resource amount were randomly generated from a predefined range. Quantity of requested/offered computational resource was varied for each request/offer message. The TTL of a request/offer message was fixed to 10000 milliseconds and reflected the delays observed in PlanetLab.

5. Experimental results

In this section, we first present the experimental results with one matchmaker and discuss how we determine the lower and upper threshold levels for one matchmaker that will be used for promotion and demotion of additional matchmakers. We then present experimental results with multiple matchmakers to show the effect of dynamically introducing and removing multiple matchmaker.

Figure-1 depicts the impact of workload on request/offer utilization. *cUtil* represent the consumer utilization and *pUtil* represent the producer utilization. As can be observed, the request/offer utilization decreases with increasing workload. As there are fewer request than offers in RIN, requests get matched as soon as they are submitted to the matchmaker and hence consumer utilization is about 100% initially. The consumer utilization keeps on decreasing with increasing workload of the matchmaker.

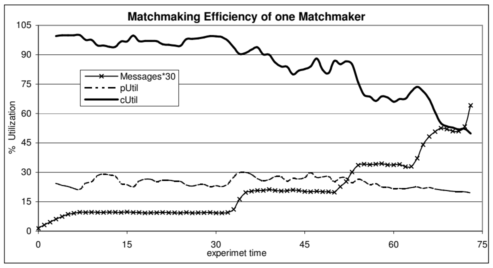

Figure 1. Matchmaking Efficiency of one matchmaker with increasing workload

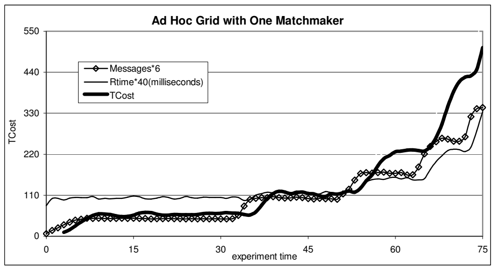

Figure 2. TCost & Response Time of one matchmaker with increasing workload

A similar evolution can be seen in Figure-2 which depicts the effect of increasing workload on the TCost and the response time. As can be expected, the TCost is proportional to an increasing workload. The response time initially remains constant and then increases with increasing workload. The constant trend is due to the fact that a request get matched as soon as it is received by the matchmaker. The increase in response time reflects the increased processing time of the matchmaker with increasing workload.

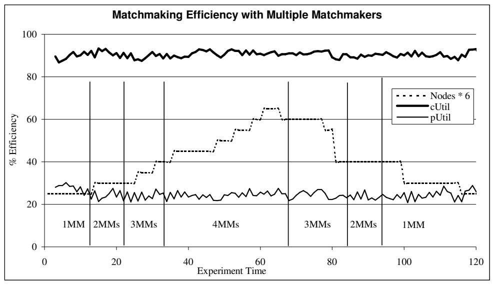

Figure 3. Matchmaking Efficiency with multiple adaptive matchmakers

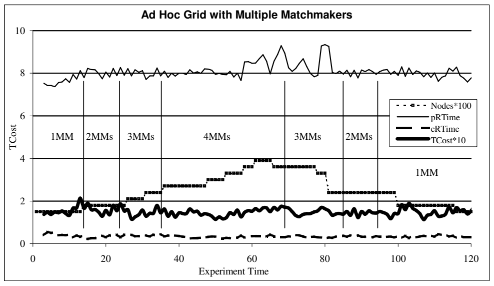

Figure 4. TCost & Response Time (milliseconds) with multiple adaptive matchmakers

From figure-1 & 2, we can observe that the matchmaker TCost and the matchmaker response time increase, whereas matchmaking efficiency decreases with increasing workload. It depicts that the matchmaker is overloaded. The matchmaker is unable to maintain its matchmaking efficiency in overloaded condition. It also implies that the consumer/producer have to pay a higher TCost for availing the matchmaking service of the matchmaker. The matchmaker needs a second (or more) matchmaker(s) to maintain

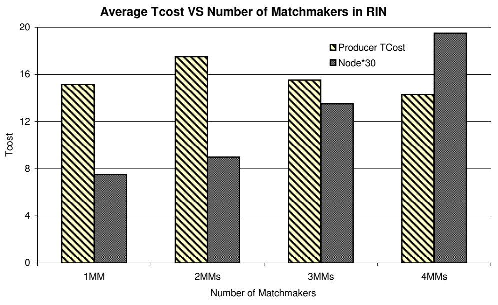

Figure 5. Average TCost and Number of Matchmakers

its matchmaking efficiency. The TCost upper threshold was 15 with the given experimental setup.

In the next set of experiments, we will show that by adding new matchmakers dynamically has a positive impact on the matchmaking efficiency and an attenuating effect on the TCost and response time. Figure-3 depicts the request/offer utilization with multiple adaptive matchmakers. The offer utilization in RIN is generally proportional to the percentage of the scarce commodity in RIN. The request utilization is about 90%. Some requests expire during the waiting period.

Figure-4 depicts the consumer/producer response time and producer TCost with multiple adaptive matchmakers. As tasks are scarce and resources are in abundance in RIN, therefore, a request is matched as soon as it is submitted to the matchmaker. Consequently, the consumer TCost is almost '0'. A new matchmaker is introduced when the first matchmaker reached its TCost threshold value.When new matchmakers are introduced, the TCost value decreases after the introduction of a new matchmaker. This phenomenon is reversed when a matchmaker is removed. Again, we can observe that the TCost remains stable irrespective of the number of messages it receives from the grid. Temporary fluctuations in the TCost also occur, reflecting variations in the grid activity. The consumer response time is much lower than the producer response time due to the abundance of resources. It is easy for a task to find the appropriate resources but the opposite is not the case. Figure-5 depicts the average TCost of number of matchmakers with increasing workload. It becomes clearer from Figure-5 that ad hoc grid can process more workload, and yet the average TCost of all matchmakers at any given instance remains closer to the upper TCost threshold for one matchmaker.

When comparing Figures-1 & 3, we can see that the matchmaking efficiency remains stable in spite of an increase of the workload. This is because the workload is now spread over 2 or more matchmakers. When comparing Figures-2 & 4, we can see that the upward pressure on the TCost has been neutralised again thanks to the change in the number of matchmakers.

In conclusion, we can observe from the above experiments that the capability of the ad hoc grid to instantiate multiple matchmakers has a stabilising effect on the TCost and response time without affecting negatively the offers/request utilization. This way, we guarantee that the transaction cost and response time become invariant to the scale on which the grid is operating.

6. Conclusions

A self-organizing mechanism, for a dynamic ad hoc grid infrastructure was proposed in this paper. The proposed mechanism focuses on the workload of the matchmaker to introduce self-organization in the ad hoc grid. The *upper* and *lower values* of matchmaker workload threshold (TCost) were determined. The upper and lower threshold values are applied to dynamically introduce multiple matchmakers such that the grid can continue to execute all its tasks irrespective of the number of messages that are being sent. All experiments were executed on PlanetLab providing a realistic platform for testing the proposed mechanisms. Future research will consist of relaxing the simplifying assumptions listed in the beginning of the paper. The most important being an extension of the underlying overlay network to manage the (dis)appearance of matchmakers in the grid and to route the messages to the appropriate matchmaker. We will look into a dynamic TCost threshold calculation mechanism that will not be bound by any specific experimental context.

References

[1] PlanetLab online, https://www.planet-lab.org/.

[2] P. Anand, W. Shaowen, G. Sukumar, and B. Ransom. A self-organized grouping (SOG) method for efficient grid resource discovery. In *Proceddings of 6th IEEE/ACM International Workshop on Grid Computing (Grid 2005)*.

[3] A. R. Butt, R. Zhang, and Y. C. Hu. A self-organizing flock of condors. *Journal of Parallel and Distributed Computing*, 66(1):145–161, 2006.

[4] S. Choi et al. Adaptive group scheduling mechanism using mobile agents in peer-to-peer grid computing environment. *Applied Intelligence*, 25(2):199–221, 2006.

[5] K. Erciyes et al. A cluster-based dynamic load balancing middleware protocol for grid. *LNCS*, 3470:805–812, 2005.

[6] R. Gupta, V. Sekhri, and A. K. Somani. Compup2p: An architecture for internet computing using peer-to-peer networks. *IEEE Transactions on Parallel and Distributed Systems*, 17(11):1306–1320, 2006.

[7] D. Hausheer and B. Stiller. Decentralized auction-based pricing with peermart. In *The Processdings of IM*, 2005.

[8] B. Pourebrahimi, K. Bertels, G. Kandru, and S. Vassiliadis. Market-based resource allocation in grid. In *2nd IEEE International conference on eScience and grid Computing*, 2006.

[9] B. Pourebrahimi, K. Bertels, S. Vassiliadis, and L. Alima. A dynamic pricing and bidding strategy for autonomous agents in grids. In *Proceedings of AP2PC*, 2007.

[10] A. Rowstron and P. Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *Proceedings of IFIP/ACM International Conference on Distributed Systems Platforms (Middleware 2001)*.

[11] D. Thain, T. Tannenbaum, and M. Livny. Distributed computing in practice: The condor experience. *Concurrency Computing.: Practice Experience*, 17:323–356, 2005.

[12] W. Yang, N. B. Abu-Ghazaleh, and M. J. Lewis. Automatic clustering for self organizing grids. In *Proceedings of IEEE International Conference on Cluster Computing*, 2006.

[13] D. Zhou and V. Lo. Wavegrid: A scalable fast-turnaround heterogeneous peer-based desktop grid system. In *Proceedings of 20th IEEE International Parallel & Distributed Processing Symposium*, 2006.