On the Feasibility of Decentralized Grid Scheduling

Marco Fiscato Paolo Costa Guillaume Pierre
Department of Computer Science
Vrije Universiteit
Amsterdam, The Netherlands
mfo300@few.vu.nl costa@cs.vu.nl gpierre@cs.vu.nl

Abstract

Many authors recognize the limitations of hierarchical Grid scheduling in scalable environments, and proposed peer-to-peer solutions to this problem. However, most peer-to-peer grid resource management systems allow only to discover available resources at the time of the request. We claim that peer-to-peer techniques have the potential for actual Grid scheduling, where each resource maintains a schedule of its future allocation to jobs. We present such a protocol, which additionally allows users to specify desired properties about the requested schedules.

1 Introduction

The landscape of utility computing is changing. The popularity of Grid and Cloud computing paradigms is leading to an increase of the size of such platforms, the heterogeneity of hardware resources and, in the case of Grid computing, the number of administrative domains. At the same time, not every organization owns a powerful cluster computer to contribute to a grid infrastructure, so we should expect next-generation Grids to be composed of dozens of thousands of small clusters and even individual machines, rather than dozens of powerful supercomputers [2].

An essential building block for such systems is job scheduling. We define scheduling as the process of selecting J machines in the infrastructure such that they will be simultaneously available to execute a job in the near future for a given duration T. In a Grid computing environment, relying on a single centralized scheduler is impossible for scalability and political reasons. Instead, most Grid platforms rely on meta-scheduling, where each cluster is given its own local scheduler, and global job scheduling is realized by delegating each job to one carefully selected cluster capable of running it [6]. However, in an environment composed of large numbers of small clusters or individual machines, scheduling jobs onto resources spanning multiple organizations becomes indispensable as no single cluster may have sufficient resources or the willingness to run a large job. This has lead a number of authors to propose peer-to-peer solutions to the Grid scheduling problem [1, 4, 7]. However, such systems focus on *discovery* of available resources rather than on resource usage *scheduling*, and the limited level of control they give to users makes their practical applicability debatable.

Our position is that peer-to-peer technologies have the potential for building full-featured decentralized Grid schedulers. To support this claim, we present an example of a peer-to-peer algorithm capable of building efficient schedules in a scalable fashion. Unlike current peer-to-peer schedulers, our algorithm plans the future allocation of tasks to resources rather than simply allocating currently available resources. It also allows users to specify an optimization criterion representing the preferences of the job with respect to the selection of resources to assign to it. We expect that many grid jobs will require their jobs to execute on nodes located close to each other, preferably in the same cluster. However, other optimization criteria may also be used. For example, the user of an interactive application may prefer his/her job to start as soon as possible, even if this implies that the selected nodes will not necessarily be colocated. Jobs that manipulate large amounts of data may prefer to execute at nodes that have good bandwidth to the location where the data are stored. Globally distributed services such as content delivery networks may require resources to be located as far away from each other as possible, or in as many different countries as possible.

The algorithm deliberately ignores the potential structure of a grid as a collection of clusters. Instead, it lets each compute node be responsible for setting up its own job execution agenda. Nodes cooperate in a gossip-based fashion in order to identify a group of nodes that can best accommodate the needs of each submitted job. A job can be submitted at any node of the system. An initial schedule is generated out of an arbitrary sample of J nodes. Of course, this schedule is unlikely to have a good quality in terms of the optimization criterion. It is therefore incrementally improved by exploring more nodes likely to have better characteristics and schedules. We show that a random selection of the extra nodes to explore provides reasonable schedules. However, the algorithm can be improved by exploiting prior knowledge of typical optimization criteria, leading to much better schedules being found in less iterations.

This paper is structured as follows. Section 2 presents related work. Then, Section 3 presents our model, and Section 4 shows preliminary performance results. Finally, Section 5 ends the paper with brief concluding remarks.

2 Related Work

A number of research efforts have recognized the need for decentralized Grid scheduling algorithms. However, to our knowledge no published algorithm simultaneously offers scalability properties and actual scheduling of resources as opposed to idle resource discovery.

One example of scalable scheduling system is Zorilla [4], which uses dissemination algorithms to locate a number of idle machines upon a job request. The dissemination infrastructure explores close-by nodes first so the resulting set of machines is due to exhibit some degree of geographical locality. However, such an algorithm can work only in an infrastructure where a sufficient fraction of the resources is idle at the time a job is submitted. In exploitation systems where the resource utilization approaches 100%, efficient resource utilization requires advance planning.

A different approach consists of letting applications be in control of the allocation of resources to them [1]. Although this approach potentially allows to fine-tune the resource selection to the precise needs of individual applications, it again relies on resource discovery rather than scheduling.

One approach similar to ours is [7], which uses genetic algorithms to build efficient schedules in an open environment. Although this approach has been shown to have very good performance in small-scale environments, it relies on global knowledge about the Grid condition. We consider that large-scale next-generation grids will make the collection of such information infeasible. Our approach, in contrast, uses similar genetic algorithms but requires only very partial knowledge about the Grid as a whole.

3 Protocol Description

Hereafter we outline our approach to build schedules in a fully decentralized fashion. For illustration purposes, after discussing our reference model, we start introducing a basic version of our protocol, which we will use as bottom line in our experiments, and then we describe how we can efficiently improve the quality of the schedules and accelerate the scheduling process, by proactively maintaining additional information about other nodes in the system.

3.1 Model

We assume that all nodes of the Grid, despite their heterogeneity, are equally capable to run any job[1]. We expect users to formulate job submission queries in the form of a triplet $\langle J, T, Opt \rangle$ where J is the requested number of nodes, T is the requested duration where these nodes should be simultaneously available, and Opt is a cost function representing the job-specific optimization criterion. The preferred schedule, out of any number of functionally correct ones, is defined as the one which minimizes function Opt. Standard optimization functions may be provided such as "find nodes as close to each other as possible" or "start the job as soon as possible." However, users may also provide custom optimization functions as a combination of several standard functions ("select nodes located as close to each other as possible, such that the job starts in at most 1 hour").

In our system, each node maintains its own schedule, defined as a list of $\langle time\_interval, job\_id \rangle$ pairs representing the reservations that this node has already committed to regarding its future utilization. We assume a model based on exclusive reservations, which means that the time intervals of two reservations may not overlap.

3.2 Basic Protocol

In our algorithm, all nodes are considered as equal so there is no designated "scheduler" nodes nor node directory. Instead, all nodes take part in a randomly structured overlay

[1] In practice, this assumption can be easily satisfied by performing a first selection of nodes based on their hardware and software attributes (e.g., using the approach described in [3]).

network, based on the CYCLON gossip protocol [9]. Every node maintains a small list of K_c random links to other nodes in the system (with $K_c \ll N$). Every node periodically selects one neighbor among these K_c nodes and exchanges a few of its links with those from its neighbor's list. This way, all nodes are periodically provided with a refreshed set of links to other randomly chosen nodes. As a consequence, the resulting overlay closely resembles a random graph. Failing nodes are quickly replaced and removed from the lists of other nodes. Such overlays have been shown to be extremely robust against partitioning, even in the presence of churn and massive node failures.

A basic, naive version of our algorithm works as follows. A job submission query can be issued at any Grid node, possibly after a number of authentication and authorization checks have been made by that node. This node recursively explores the CYCLON until it discovers $\alpha \times J$ nodes, where α is a parameter of the algorithm[2]. A first schedule can be generated by selecting J nodes out of $\alpha \times J$ such that the optimization function Opt is minimized.

In practice, exploring all permutations of J nodes out of $\alpha \times J$ is computationally infeasible unless $\alpha \approx 1$ or J is very small. However, having $\alpha \gg 1$ is essential to identify good schedules, as it gives the algorithm more opportunity to ignore unsuitable nodes. To this end, we implemented the selection process using a genetic algorithm [5] to obtain nearly optimal results in polynomial time.

The overall quality of the resulting schedules, however, strictly depends on the random nodes found in the CYCLON caches. To minimize the impact of randomness and to increase the chances to find better schedules, the above process is iterated a number of times, stopping either after τ iterations or at the time when the current best schedule is due to start. Each iteration is computed out of nodes found starting from a different node of the CYCLON overlay[3]. Each time a new better tentative schedule is found, the old one is discarded, the resources of the previous one are released and the new ones locked via 2-phase commit.

3.3 Proactive Protocol

While the above protocol is effective in providing correct schedules, it suffers from one major drawback: since nodes are selected on a random basis, there is a high chance that even the best set found will barely match the user expectations. For example, if we consider the proximity optimization criterion ("nodes located close to each other, preferably in the same cluster"), it is easy to see how unlikely is to find good candidates if nodes are selected randomly. The schedule obtained is functionally correct but its quality will generally be far from the optimum.

Clearly, the bigger the system is, the more critical this issue will be because the chances to find the optimal schedule will decrease at large scale. Increasing the value of τ and α may improve the quality of the solutions, but at the cost of increasing the computational overhead.

We alleviate this problem by constructing extra overlays on top of CYCLON. While CYCLON aims at maintaining a random graph, these extra overlays aim at linking together nodes that are likely to produce good schedules if allocated together. Multiple such overlays can be built to cluster nodes according to different metrics. For example, to support queries for nodes located close to each other, we should build an overlay that links nearby nodes together. To support scheduling requests for jobs to start as soon as possible, we should build an overlay that links nodes whose first available time slots largely overlap.

Building and maintaining such "helper overlays" that link together nodes which share a certain characteristic can be realized using a second gossip layer, called VICINITY [10], on top of the CYCLON overlay network. Similarly to the CYCLON protocol, VICINITY keeps another set of K_v links to other nodes, and periodically exchanges information about a subset of its links K_v with its neighbors. Differently from CYCLON, however, in VICINITY nodes do not randomly select links to keep in their list but always keep the best according to the overlay metric (e.g., proximity or time of the first available slot). One may thus construct custom overlays on demand by simply defining a distance function representing the metric whose value should be optimized. One may build an overlay linking nodes with largely overlapping first available time slot by using the size of this overlap as the optimization criterion. Similarly, one may link nearby nodes by using the inter-node latency, for example as predicted by a network coordinate system [8]. Note that maintaining several such overlays does not imply major overhead: each overlay generates a constant background traffic in the order of a few hundred bytes per second.

The maintenance of helper overlays can be considered as a proactive, though lightweight, way to to maintain sets of nodes likely to produce good schedules if used together. When a query is issued, at each iteration the node processing the query can select one (random) node of out its CY-

[2]In our experiments we find that $\alpha = 2$ exhibits a good tradeoff in terms of cost and performance.

[3]Note that since iterations are completely independent from each other, they can be run in parallel by concurrent threads to improve performance.

CLON cache and $\alpha \times J - 1$ nodes out of that random node's relevant helper overlay. For instance, if proximity among nodes is required, then each iteration will evaluate $\alpha \times J$ nodes located close to a randomly selected node. Using helper overlays as a lightweight way to group nodes that share certain characteristics significantly improves the quality of the obtained schedules and reduces the number of necessary iterations, as shown in the next section.

4 Evaluation

To assess the performance of our protocol we emulate a whole Grid system by running N instances of our implementation on a single machine. We study the cases $N = 800$ and $N = 10,000$. Due to space constraints, we focus only on the proximity metric ("find nodes as close to each other as possible") as this is arguably required by many grid application developers. Nevertheless, similar trends are also exhibited by the other functions we tested.

To replicate a realistic situation, we assigned each emulated node with the network coordinates of a real node positioned during a previous study [8]. Latency between two nodes is estimated as the Euclidean distance between their respective coordinates. Note that this setup is more challenging than most Grid systems, in that it is composed only of individual machines rather than entire clusters.

To measure the quality of the solution found, we define the optimization function as: $Opt(n_1, n_2, \ldots, n_J) = \sum_{i=1}^{J} d(n_i, \bar{n})$ where $d(a, b)$ is the Euclidean distance between the coordinates of node a and node b and $\bar{n}$ represents the geometric center of the set of nodes $\{n_1, n_2, \ldots, n_J\}$. The goal of our protocol is to minimize this function.

Figure 1 plots the value of the optimization function related to the best schedule at a given iteration using respectively the basic protocol and the proactive one, in four different scenarios requesting $J = 64$ (resp. $J = 128$) nodes among a Grid of $N = 800$ (resp. $N = 10,000$) nodes. The schedule performances are expressed as the quality ratio versus an "Centralized" schedule obtained by running our genetic algorithm through the entire set of nodes, as a centralized scheduler would.

In all cases, the proactive protocol clearly outperforms the basic one both in terms of the quality of the schedule found and in terms of required number of iterations. Quality-wise, the schedules found by the proactive protocol closely approach those of the centralized scheduler. The proactive protocol also finds good schedules in very few iterations. By exploiting the information provided in the upper layer overlay, it can immediately start with a reasonably good schedule and then improve on it by exploring other close groups of nodes.

Arguably, the proactive protocol improves the scheduling performance at the cost of extra network traffic. Each metric that users may want to optimize against requires a specialize overlay. The implied traffic, however, remains extremely low, with each node exchanging around 1,280 bytes per proactive overlay at every gossip cycle. Given gossip periodicities around 10 seconds, we consider these costs as negligible.

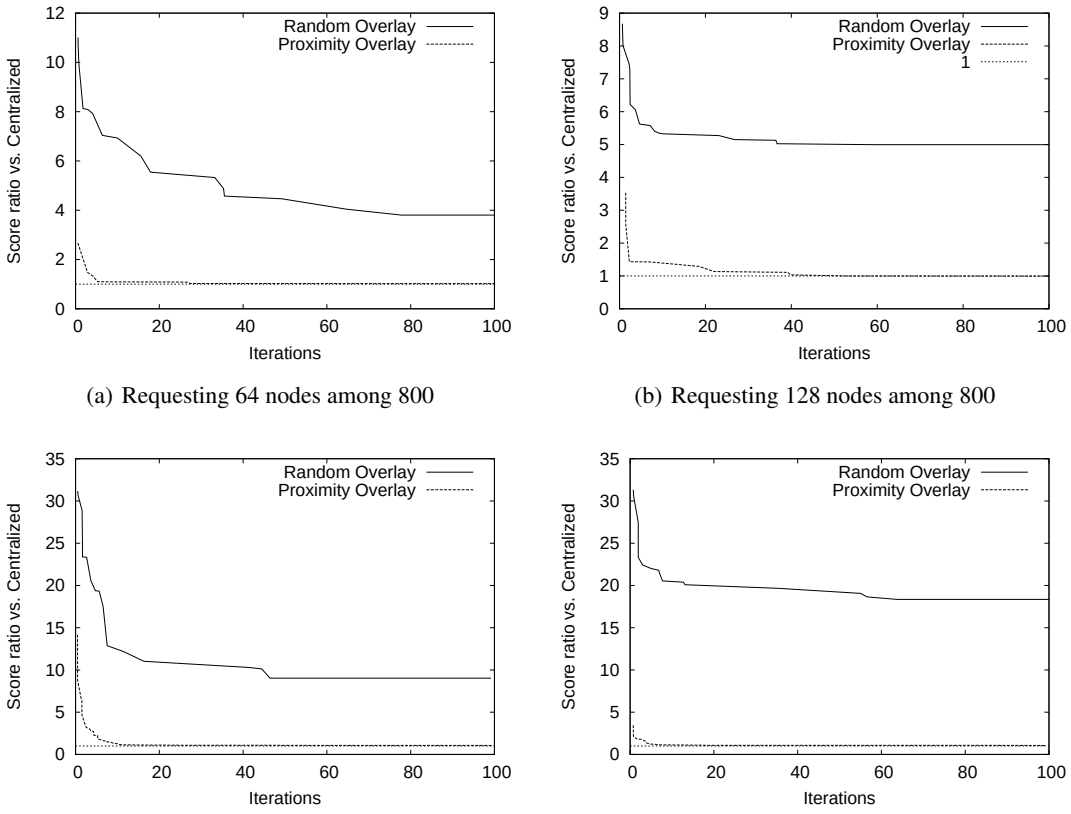
5 Conclusions

The recent trends in distributed computing are leading to a scenario in which many organizations will collaborate to offer virtually unlimited resources to run users' jobs. Traditional scheduling approaches are inadequate to cope with the new scalability challenges that this scenario brings along and many researchers agree on the need to explore decentralized and autonomous solutions.

To support this claim, as a proof-of-concept, we designed and evaluated a fully decentralized protocol to schedule jobs in a large-scale multi-domain system. The protocol relies on gossiping to evenly spread the overhead across the entire set of nodes and to ensure high robustness, thus preventing bottlenecks or single points of failures typical of centralized systems. Differently from existing peer-to-peer solutions, our protocol empowers users with fine-grained control over the resource allocation by allowing for job-specific optimization criteria. It also enables planning the allocation of resources in the future and not only at the time of the request.

Despite the encouraging results, we are well aware that this is just a preliminary step towards the development of a fully-fledged decentralized scheduling solution. Several key features are still lacking like an effective charging model and the support for execution policies (e.g., "user from organization X cannot run jobs on hosts of company Y"). Especially policies nowadays play a major role in mainstream systems and their relevance is likely to grow in the future due to the explosion of administrative domains. Beside, another missing functionality is a decentralized monitoring service to track job executions and to timely replace failed nodes.

Investigating these issues as well as running a large-scale deployment of our protocol is in our immediate research agenda.



(a) Requesting 64 nodes among 800

(b) Requesting 128 nodes among 800

(c) Requesting 64 nodes among 10,000

(d) Requesting 128 nodes among 10,000

Figure 1. Quality of the schedule found against the number of iterations

Acknowledgements

This work has been funded by the XtreemOS FP6 project [2].

References

[1] A. Chakravarti, G. Baumgartner, and M. Lauria. The Organic Grid: Self-Organizing Computation on a Peer-to-Peer Network. In *IEEE Transactions on Systems, Man, and Cybernetics*, volume 35, May 2005.

[2] T. Cortes, C. Franke, Y. Jégou, T. Kielmann, D. Laforenza, B. Matthews, C. Morin, L. P. Prieto, and A. Reinefeld. XtreemOS: a Vision for a Grid Operating System. XtreemOS technical report #4, May 2008. `www.xtreemos.eu`.

[3] P. Costa, G. Pierre, and M. van Steen. Autonomous Resource Selection for Utility Computing. Submitted for publication, 2008.

[4] N. Drost, R. V. van Nieuwpoort, and H. E. Bal. Simple Locality-Aware Co-allocation in Peer-to-Peer Supercomputing. In *Proc. Intl. Workshop on Global and Peer-2-Peer Computing*, May 2006.

[5] A. Eiben and J. Smith. *Introduction to Evolutionary Computing*. Natural Computing Series. Springer, 2003.

[6] E. Huedo, R. Montero, and I. Llorente. The GridWay Framework for Adaptive Scheduling and Execution on Grids. *Scalable Computing - Practice and Experience*, 6(3):1–8, 2005.

[7] G. Iordache, M. Boboila, F. Pop, C. Stratan, and V. Cristea. A Decentralized Strategy for Genetic Scheduling in Heterogeneous Environments. In *Proc. Intl. Conf. on Grid Computing, High-Performance and Distributed Applications*, 2006.

[8] M. Szymaniak, D. Presotto, G. Pierre, and M. van Steen. Practical Large-Scale Latency Estimation. *Elsevier Computer Networks*, 52(7):1343–1364, May 2008.

[9] S. Voulgaris, D. Gavidia, and M. van Steen. CYCLON: Inexpensive Membership Management for Unstructured P2P Overlays. *Journal of Network and Systems Management*, 13(2), 2005.

[10] S. Voulgaris and M. van Steen. Epidemic-Style Management of Semantic Overlays for Content-Based Searching. In *Proc. Euro-Par Conf.*, Aug. 2005.