ozmos: Bio-Inspired Load Balancing in a Chord-based P2P Grid

Amos Brocco
Institute of Telematics, Karlsruhe Institute of Technology (KIT) Zirkel 2, D-76128 Karlsruhe, Germany
brocco@tm.uka.de

ABSTRACT

Load balancing in distributed computing systems is an important requirement to make efficient use of all available resources. Envisioning a increase in the scale and dynamicity of future grid systems, fully distributed autonomic solutions are required to address this problem. In this regard, we introduce a load balancing mechanism, called OZMOS, that follows the principle of osmosis to relocate tasks between nodes in a P2P based grid. Our solution is based on a Chord overlay upon which bio-inspired agents are deployed to share information about the status of the grid as well as to reschedule tasks between nodes. The key based routing capabilities of Chord are exploited to discover other nodes in the overlay, and to efficiently support relocation of incompatible tasks in heterogeneous grids. By means of a simulation study conducted in various scenarios, we highlight the efficacy of the proposed algorithm in achieving system-wide load balance in grids of different scales, and with both homogenous and heterogeneous resources.

Categories and Subject Descriptors

C.2.4 [**Computer-Communication Networks**]: Distributed Systems; I.2.8 [**Problem Solving, Control Methods, and Search**]: Scheduling

General Terms

Algorithms, Performance, Experimentation

Keywords

Load Balancing, Grid, P2P, Bio-inspired Algorithms

1. INTRODUCTION

The convergence of grid and peer-to-peer systems [1] has introduced several opportunities and new challenges in the realm of grid computing. On one hand, this trend has opened up the possibility of building distributed computing systems with smaller investments for the underlying infrastructure, and with considerably larger scales. On the other hand, the assumptions about dependability, efficiency and security of traditional grids have been voided due to the inherent dynamic nature of the P2P paradigm. As a consequence, research has focused on exploiting the benefits of decentralization and peer-to-peer solutions while trying to solve the aforementioned issues.

Effective grid computing relies on efficient allocation of tasks on the most appropriate resources [2]. To achieve that, schedulers in traditional centralized grids exploit full knowledge about the status of resources to optimally match user demands (i.e. QoS agreements such as response time, price, etc.) with availability and usage policies set by resource providers (i.e. security, resource utilization, etc.). While such a centralized model is imposed by the business requirements set by virtual organizations, research has focused on solutions that reduce deployment costs, increase reliability, and meet dynamic users' needs, envisioning flexible, autonomic, and self-manageable grids [3]. Even in the light of the recent move toward *cloud* platforms, which have evolved from grids and typically rely on a grid computing infrastructure, problems such as adaptive scheduling and dynamic distribution of loads remain of great interest. In fact, as hinted in [4], in order to support on-demand provisioning, both grid and cloud computing infrastructures are likely to become large-scale heterogeneous platforms with small providers coexisting with larger ones. This shift will inherently involve the development of new mechanisms for providing fully distributed efficient task allocation. To tackle this problem, in this paper we present a fully distributed task load balancing mechanism that employs bio-inspired techniques to provide autonomous and self-organized operation. More specifically, our solution is based on ant inspired agents that reorganize tasks among the available resources following the principle of osmosis. Osmosis is a self-regulatory process where a solvent moves across a semi-permeable membrane separating two solutions with different solute concentration. The algorithm, named OZMOS, is based on a variation of this process (working the opposite way) that is executed on a Chord [5] overlay to optimally relocate jobs between nodes both in homogeneous and in heterogeneous grid systems. The remaining of this paper is organized as follows: Section 2 discusses related work concerning distributed task load balancing with a focus on bio-inspired approaches. Section 3 presents the proposed load balancing solution, while Section 4 and 5 present the considered evaluation scenarios and discuss the respective results. Finally Section 6 summarizes our conclusions on this work and provides some insights on future work.

2. RELATED WORK

Resource management and scheduling middlewares for grid computing deal with large pools of dispersed resources; despite such great availability, it is still important to make efficient use of computing power, namely by assigning jobs to the most appropriate nodes and by avoiding to overload just some systems. Accordingly, in this section we present a brief review of the existing rich literature concerning decentralized scheduling and load balancing in grids. Job scheduling in grids is managed by meta-schedulers, which consider all available resources as a whole, and operate over the existing local scheduling policies of each node. Traditional grid solutions, such as [6, 7], employ centralized or hierarchical meta-schedulers that can exploit complete knowledge about the available resources, enabling optimal scheduling decisions. Unfortunately, several issues derive from these designs, such as scalability bottlenecks, and single points of failure that affect the robustness of the whole grid. In this regard, there has been a great effort in researching distributed scheduling solutions that can overcome the flaws of centralized systems, while still providing optimal scheduling with minimal communication overhead [8]. Furthermore, load balancing can be performed either statically or dynamically [9]: whereas dynamic systems employ live information to balance the system load at run time, static load balancing solutions assume that the characteristics of the resources are known a-priori and do not change during execution. Such assumptions are difficult to ensure in fully distributed systems [10], and the focus of our research is thus on decentralized dynamic load balancing.

The load balancing problem is closely related to scheduling, hence several distributed scheduling frameworks already include load balancing capabilities; examples include [11], which introduces a fully distributed meta-scheduling protocol based on an cost-for-execution heuristic, [12], where a node retains jobs or reschedules them to its neighbors according to the actual load, and [12], where each node employs an heuristic based on local load to decide if a job is to be delegated to a neighbor. In [13] an adaptive load balancing strategy for a super-peer based P2P grid is detailed, with super-peers managing the resources of other nodes and exchanging information among each other. For each job, super-peers determine the estimated completion times for each of the neighboring nodes, and eventually migrate the job to the most appropriate one. The solution presented in [14] employs decentralized and adaptive scheduling and load balancing strategies to balance total execution time across nodes. A similar direction is followed by [15], which proposes an adaptive decentralized algorithm based on evolutionary techniques to delegate job execution. Another example, presented in [16], discusses a P2P computing system based on a resource trading model is presented. The proposed solution replaces centralized resource management and job submission with a distributed matchmaking process that mimics the operation of a trade market.

In order to overcome some of the complexity issues, bio-inspired techniques are increasingly being considered. For instance, in [17] the grid is considered as a self-organized complex system where agents are used to provide decentralized task scheduling. Nodes are organized hierachically, and achieve load balancing by transferring jobs according to the actual load of their neighbors. A notable example of load balancing mechanism based on artificial life behaviors is Messor [18], which builds upon the swarm intelligence [19] and ant colony [20] techniques. Messor employs ant-like agents whose behavior is determined by the actual load of a node. If the origin node is overloaded, the agent's goal is to find an underloaded node; on the contrary, the agent will look for overloaded systems. Each agent wanders on the network up until a node with the desired load is found and the balancing can take place. Furthermore, agents collect and share information about the load of visited nodes, in order to direct subsequent agents toward nodes of major interest. The algorithm proposed by Messor has inspired the adoption of similar techniques in other platforms such as ARMS [21], and [22]. In particular, ARMS employs an swarm based solution coupled with a multi-agent resource management system, where ants are exchanged by hierachically organized agents attached to local schedulers. A similar framework for distributed job scheduling is presented in [23]: a multi-agent design is employed both for managing local grid nodes and for discovering remote resources. Ant-like agents are deployed upon submission of a task, and wander on the network looking for suitable resources to reduce both the overall makespan (the maximum time required to complete all jobs) as well as the response time. In [24] two distributed swarm intelligence based scheduling algorithms are proposed. The first is based on the ant colony heuristic and, similarly to Messor, employs agents that wander on the network searching for the node which is best suited to carry out a job. Ant agents leave pheromone trails on their paths to indicate the fitness of each inspected resource. The second algorithm implements a distributed version of the particle swarm optimization paradigm [25], with jobs being transferred toward nodes with the highest fitness. In the literature it is also possible to find some examples of synergies between traditional P2P systems and bio-inspired solutions tackling the problem of grid scheduling and load balancing. For example, in [26] a complete meta-scheduling architecture based on ant-like agents executing on a DHT overlay is presented; like in Messor, load balancing is provided by the collaborative behavior of swarm agents. The structured overlay is used to maintain multicast communication trees and efficiently track submitted jobs.

Following similar concepts, we propose a system based on bio-inspired agents wandering on a structured P2P overlay. In contrast to the aforementioned Messor examples, our solution relies on simple interactions between adjacent nodes and the structure of the P2P overlay, and does not depend on agent-based resource discovery mechanisms. Furthermore the key-based routing is employed to quickly move agents between nodes, whereas the overlay management protocol is exploited to organize and group nodes according to their capabilities in order to efficiently support both homogenous and heterogeneous tasks and resources.

3. OZMOS PROTOCOL

The OZMOS algorithm dynamically reallocates tasks among a set of distributed resources in order to balance the overall load. The balancing process is completely decentralized, and is supported by the activity of ant-like agents that are executed by nodes on a Chord overlay. Each node manages a batch scheduling queue and can execute one or more tasks concurrently with different performance characteristics. We assume that tasks are independent, non-divisible, and non-preemptive. Our solution supports both homogeneous and heterogeneous distributed computing: in a fully homogeneous scenario all tasks have the same requirements, and each node is expected to be able to execute any task (with the same run time performance). Conversely, in a fully heterogeneous grid each task has different requirements that can be fulfilled only by some of the nodes, tasks have varying run times, and each resource has different performance characteristics. In the following we detail the operation of OZMOS for both homogeneous and heterogeneous grids.

3.1 Chord

Chord [5] is a P2P overlay that implements a distributed hashtable (DHT). The overlay is structured as a ring, and each node is assigned a unique identifier. Within the ring, nodes are ordered according to their identifier, and store pointers to their successors and to their predecessor; furthermore, each node maintains a finger table with addresses of several remote nodes to quickly forward messages toward distant regions of the network. As a distributed storage system, data is put on the node whose identifier is the closest to that assigned to the data. In an overlay of size N, Chord is able to route a message to destination in $O(logN)$ hops.

3.2 Local and remote concentrations

The load-balancing problem is concerned with distributing tasks among the available resources so that each node is equally loaded. Because we consider a network where each node has different performance characteristics, we describe the actual load of each node with a *concentration* value, which represents the time required to process the whole scheduling queue. The concentration c_M on a node M is computed according to the following formula:

$$c_M = \frac{\sum_{j \in T} j_{ert}}{speed_M \times cpu_M}$$

where T is the set of all scheduled jobs, j_{ert} is the *estimated running time* of task j in seconds (or remaining time for running jobs), $speed_M$ is the speed index of the node and cpu_M is the maximum number tasks that can be concurrently executed. We assume that a node with baseline performance $speed_M = 1$ is used to determine j_{ert} (for example by means of a profiling tool such as PACE [27]); generally a node with $speed_M = t_M$ will complete jobs t_M times faster than the baseline system. We further define the *performance index* of a node M as $perf_M = speed_M \times cpu_M$, and its *normalized concentration* as $\dot{c}_M = c \times perf_M$.

Nodes determine their load balancing actions by comparing the local concentration with that of nodes in the network. Accordingly each node receives information from its predecessor (p) and successor (s) on the ring, as well as from a *random* remote node called *probe* (r). For each $N \in \{p, s, r\}$, we assume that local values for $\dot{c}_N$ (initially set to ∞) and $perf_N$ (initialized as 1) are available. The predecessor and successor concentration values provide information about the local load balancing state, whereas the probe offers a more global overview. Because the p and s links match the underlying ring structure, they are relatively stable; meanwhile the probe link is periodically updated and is thus more volatile.

3.3 Resource identifiers

In homogeneous grids, each node is assigned a random unique identifier of m bits (typically $m = 160$) as in Chord. To deal with heterogeneous resources we restrict ourselves to a finite number of classes $s \geq 0$, and we segment the key space into s different classes. The identifier of each node is generated so that both the class identifier as well as a randomized key are encoded. Given $s = 2^k$ as the maximum number of classes, the k highest-order bits of the Chord identifier are replaced by the value of the class (Figure 1). As a result, nodes belonging to the same class connect to adjacent positions in the ring. Tasks are similarly assigned unique identifiers whose prefix matches the class of resources required for their execution. In the present implementation of OZMOS we have not considered the case of overlapping classes. Nonetheless, minor adjustments to the algorithm can be introduced to allow load balancing between diverse but overlapping classes if those map on adjacent regions in the ring.

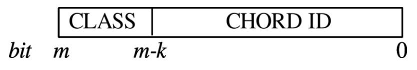

Figure 1: Node identifier with class field

3.4 Agents

The goal of each node is to minimize the distance between its local concentration and that of its neighbors by offloading tasks to other nodes in the grid. To achieve this, OZMOS employs three types of agents with different behaviors. Agents have access to the local concentration values, to the scheduling queue, as well as to Chord data structures such as the address of the predecessor, the successor list, and the finger table. In this respect, we don't currently address security issues, notwithstanding that these aspects are of great importance in grid computing. We are aware that solutions that can be deployed into real-world grids must take care of the problem, for example by verifying the origin of agents and tasks. Furthermore, our protocol is currently not concerned with recovery from node or network failures, hence queued and executing tasks would be lost if a node fails. Finally, for the actual job migration we assume that an underlying data transfer mechanism is available.

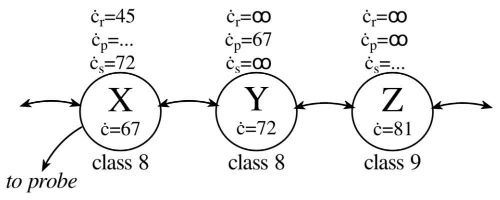

Figure 2: Example of heterogeneous grid (performance indices omitted)

Notification.

The task of *Notification* agents is to inform both ring neighbors and a remote node about the local concentration value and performance characteristics. Nodes periodically send a *Notification* agent to their predecessors, successors, and to randomly chosen nodes. Whereas the predecessor and successor addresses are known, the random node's address is drawn from the local finger table and the successors' list. To support load balancing in heterogeneous grids, updates don't take place if the target's resource class is different than that of the origin node, which means that the corresponding remote concentration value will remain equal to ∞. This condition can be verified by checking the target's identifier before transmitting the agent (in the case of successors and predecessors), and by choosing the random node only within the admissible identifiers. As shown in Algorithm 1, the agent is given a target node to migrate to (either p, s or the probe node r). The agent reads the local normalized concentration and performance index values and subsequently migrates to the target node, where the corresponding information is updated. On the target node, if the source address corresponds neither to the predecessor, nor to the successor, the probe information is updated. The activity of *Notification* agents is thus twofold: on one hand it ensures that adjacent nodes on the ring know the concentration of their respective neighbors; on the other hand it provides each node with the address of a *random* peer that represents an additional possibility for off-loading tasks to. Figure 2 depicts an example of heterogeneous grid. Nodes X and Y belong to the same resource class 8, and thus share their actual local concentration values. On the contrary, Y will not receive updates from Z, hence the concentration value for its successor will remain equal to ∞. Finally, node X has a valid *probe* address, and has thus received the corresponding concentration value $\dot{c}_r$.

Algorithm 1 Notification Agent

```
Let: $this$, the source node;
Let: $target$, the target node (either $p$, $s$, or a random one);
Let: $migrate(t)$, function to migrate to node $t$;
 1: $x := this$
 2: $c := \dot{c}_{this}$
 3: $v := perf_{this}$
 4: $migrate(target)$
 5: if $x = s$ then
 6:     $\dot{c}_s := c$
 7:     $perf_s := v$
 8: else
 9:     if $x = p$ then
10:         $\dot{c}_p := c$
11:         $perf_p := v$
12:     else
13:         $r := c$
14:         $\dot{c}_r := c$
15:         $perf_r := v$
16:     end if
17: end if
```

Osmosis.

Osmosis agents are used to migrate tasks from a node with high concentration toward a node with low concentration. Each node M periodically determines the osmotic pressure toward other nodes, by computing the difference between the local concentration and that of each node $n \in \{p, s, r\}$, namely $c_M - \frac{\dot{c}_n}{perf_n}$. The node x corresponding to the highest positive difference is chosen as candidate for the load balancing process. In order to level local concentrations, the node tries to reschedule some of the local jobs on the remote node. However, because each task represents a discrete indivisible entity, precise load balancing might not be possible. Furthermore, the load balancing operation must consider the performance of both nodes. As a consequence, a node first computes the total estimated run time that must be transferred to x as:

$$T_{ert \to x} = \frac{(\dot{c}_M \times perf_M) - (\dot{c}_x \times perf_x)}{perf_M + perf_x}$$

A set of tasks J such that $\sum_{j \in J} j_{ert} \approx T_{ert \to x}$ is selected within the local scheduling queue; tasks in set J are migrated toward node x with the following probability:

$$P_{J \to x} = min(1, 1 - (\frac{\sum_{j \in J} j_{ert} - T_{ert \to x}}{\epsilon \times T_{ert \to x}}))$$

where ϵ is a user-defined threshold. The migration process is carried out by *Osmosis* agents, whose behavior is detailed in Algorithm 2. The agent is given the list of task descriptors and a direction to follow in the ring. When the agent arrives on the target node the local concentration is checked: if this value is lower than that of the succeeding one (according to the actual traveling direction), the tasks carried by the agent are dropped and locally scheduled. On the contrary, the agent can migrate for a number of steps further in the ring. This behavior ensures that tasks are released on the least loaded node, hence improving the load balancing result. In a heterogeneous system forwarding terminates when the successor of the current node belongs to a different resource class, because the reported concentration would be infinite. If the target is the *probe* node, the traveling direction on the ring is determined by following the lowest concentration after the initial migration. The operation of the *Osmosis* agent can be viewed as the behavior of an ant following a pheromone trail, with lower concentrations being preferably chosen.

Algorithm 2 Osmosis Agent

```
Input: $J$, set of tasks to be migrated;
Input: $dir$, direction of movement (either $\rightarrow p$, $\rightarrow s$, $\rightarrow r$);
Input: $steps$, maximum number of allowed hops in the ring;
Let: $migrate(t)$, function to migrate to node $t$;
Let: $this$, the current node;
Let: $next(d)$, the following node in the $d$ direction;
Let: $schedule(t)$, schedules task $t$ on the current node;

 1: $migrate(next(dir))$
 2: if $dir = \rightarrow r$ then
 3:     if $\frac{\dot{c}_{next(\rightarrow p)}}{perf_{next(\rightarrow p)}} > \frac{\dot{c}_{next(\rightarrow s)}}{perf_{next(\rightarrow s)}}$ then
 4:         $dir := \rightarrow p$
 5:     else
 6:         $dir := \rightarrow p$
 7:     end if
 8: end if
 9: while $steps > 0$ do
10:     $steps := steps - 1$
11:     if $c_{this} \geq \frac{\dot{c}_{next(dir)}}{perf_{next(dir)}}$ then
12:         $migrate(next(dir))$
13:     else
14:         $break$
15:     end if
16: end while
17: for all $task \in J$ do
18:     $schedule(task)$
19: end for
```

Relocation.

Algorithm 3 Relocation Agent

```
Input: $R$, list of tasks to be relocated;
Input: $tclass$, resources' class of the tasks to be relocated;
Let: $this$, the current node;
Let: $route(c)$, migrate to the first node in class $c$ with
    Chord's key based routing;
Let: $class(t)$, return the resources' class of task $t$;
Let: $schedule(t)$, schedule a task $t$ on the current node;

1: $route(tclass)$
2: for all $task \in tasks$ do
3:     $schedule(task)$
4: end for
```

In a heterogeneous grid, tasks might be submitted to a node whose resource profile does not fulfill the requirement for execution. Accordingly, a relocation mechanism is required to reschedule incompatible tasks on nodes of the appropriate class. The *Relocation* agent detailed in Algorithm 3 is generated by nodes in order to dispose such jobs by migrating them into another scheduling queue. In contrast to the *Osmosis* agent, the *Relocation* one employs key based routing of the underlying Chord to directly jump to a node whose class is compatible with the tasks being relocated.

4. EVALUATION

To validate the load balancing performance of OZMOS, several experimental evaluation scenarios, with both homogeneous and heterogeneous resources, have been considered. To take into account the most prominent characteristics of the load balancing problem, different aspects are analyzed: convergence rate, scalability, stability, and network overhead. All experiments are conducted using the OverSim [28] platform, which enables faithful simulation of the underlying network characteristics. We assume that agents relocate only task descriptors, hence our simulations do not take into account the actual task data transfer. In this section the details of the simulation setup are presented and discussed.

General setup.

A total of 5 simulation runs are performed for each experiment, with each run covering 3 hours of activity. Unless otherwise stated, each experiment is conducted on grid consisting of 512 nodes bound to a Chord P2P overlay. At the beginning of each experiment, nodes join the overlay with a frequency of 1 node every second. Tasks are submitted once all nodes are connected to the overlay. In all experiments the algorithm parameters have been kept constant. The osmosis threshold ϵ is set to 1.05, which means that load balancing occurs with probability < 1 as soon as the total run time length of the tasks to be migrated exceeds by 105% the requested amount. Nodes notify their neighbors about the local concentration and performance every 30 seconds on average. Osmosis is performed (when necessary) with a frequency of one every 60 seconds. Relocation of incompatible jobs is performed every 120 seconds on average: only one class of incompatible jobs is relocated at a time. A detailed sensitivity analysis of these parameters has not yet been carried out, and is left as future research work. All nodes communicate asynchronously, and throughout all experiments we assume that the network is reliable and does not experience neither communication nor node failures. As we are only interested in the load balancing of scheduling queues, job execution is not considered in the simulation. Finally, *Osmosis* agents can travel at most 10 hops in the overlay looking for lower concentration nodes.

Full homogeneous scenarios.

In homogeneous scenarios all nodes are grouped into a single resource class, and hence have equal characteristics and are able to execute any of the submitted task with a performance of at most 1 task at a time ($cpu_M = 1$) with $speed_M = 1$. One node is given 50000 tasks at the beginning of the simulation, with a run time j_{ert} of 45 minutes for each task. For scalability testing, experiments with 10000 tasks are also conducted.

Full heterogeneous scenarios.

In heterogeneous scenarios both nodes and tasks are attributed a random resource profile out of the 16 available classes. Computing power of each node is randomly assigned: for each node, cpu_M is uniformly chosen on the discrete interval $[1, 4]$, whereas $speed_M$ varies on a continuous interval between 1 and 2. Task run times j_{ert} are chosen uniformly at random in the continuous interval between 30 minutes to 1 hour. In each simulation run, 50000 jobs are submitted to a random node (10000 in scalability experiments).

Mixed scenarios.

To evaluate the influence of each factor of heterogeneity (either different task run time, $perf_M$, or resource class) several intermediate scenarios are considered. Starting from the full homogeneous settings, the performance of each machine is changed by assigning a random $speed_M$ and cpu_M to each node as in the heterogeneous setup. This configuration bears similarities with typical desktop grid applications like Folding@home [1] or SETI@home [2], where work units submitted to each node have equal size and are of similar complexity, but the actual run time depends on the hardware configuration of each computer. A second scenario involves experiments with varying task run times scheduled on homogeneous hardware. Finally, load balancing of tasks with equal j_{ert} on nodes with different resource classes but homogeneous performance is evaluated.

Measurements.

The load balancing performance of the algorithm is determined by the relative standard deviation (%RSD) of the concentration. In homogeneous scenarios this value is computed over all nodes, whereas in heterogeneous scenarios separate data is computed for each resource class and only the maximum %RSD is considered. Regarding the stability of the algorithm, the number of tasks that are relocated every 30 seconds is examined; this value also hints the amount of traffic generated by the algorithm in terms of task descriptors transmitted over the network. We also evaluate the scalability of the algorithm in relation to two different axes: overlay size and number of tasks. Concerning the first, results obtained in overlays of 256, 512, and 1024 nodes are compared, whereas for the second, load balancing of 10000 and 50000 tasks are examined.

[1]http://folding.stanford.edu/

[2]http://setiathome.ssl.berkeley.edu/

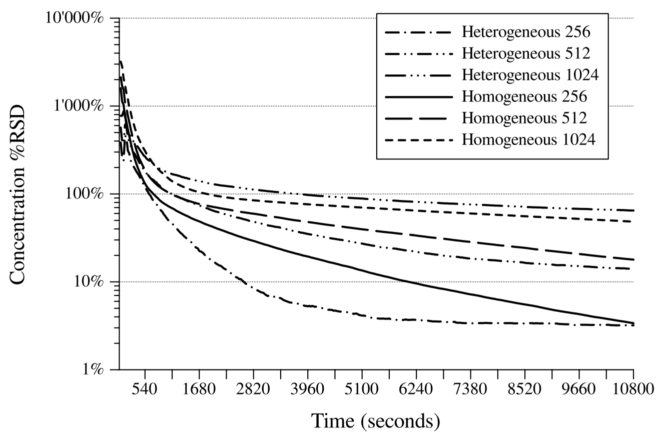

Figure 3: Load balancing performance / Scalability (overlay size)

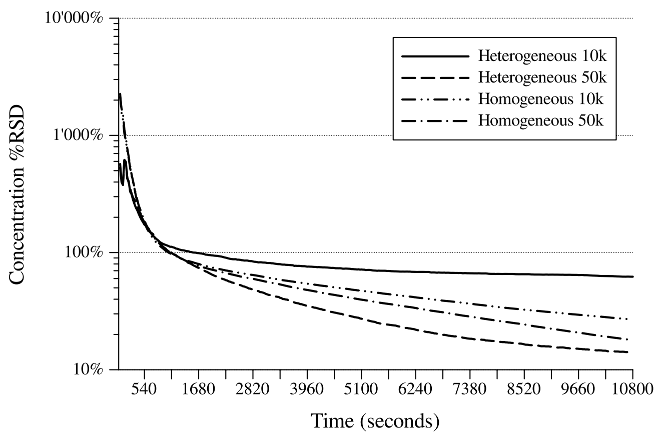

Figure 4: Scalability (task count)

5. RESULTS

In this section we present and analyze the results of the previously discussed experiments. All the reported values refer to an average across 5 simulation runs.

Load balancing performance.

The first set of results concerns both homogeneous and heterogeneous scenarios with 50000 tasks, and aims at evaluating the evolution of the relative standard deviation throughout each simulation at different overlay scales (256, 512, and 1024 nodes). As shown in Figure 3, the %RSD rapidly converges below 100% in all scenarios. It is interesting to note that heterogeneous scenarios obtain better results, since the number of nodes and tasks in each class is lower than in the homogenous setup; more specifically, each class counts an average of 3125 jobs as well as 16, 32, and 64 nodes in overlays of size 256, 512, and 1024 respectively. This scale difference allows for a faster relocation of tasks among all compatible resources. Although not shown in the graph, in the heterogeneous scenario an average of only 15 *Relocation* agents are necessary to reschedule all incompatible tasks (about 47000) to a node in the appropriate resource class. It should be noted that this result is due to relocations being carried out early in the simulation by the lone node carrying the initial load.

Task count scalability.

Figure 4 illustrates the scalability of the system concerning the number of tasks in both homogeneous and heterogeneous setups. In all experiments the size of the network is 512 nodes. Surprisingly, a higher number of tasks improves the convergence of the concentration's %RSD: in homogeneous scenarios the difference is very slight (17.9% versus 26.8%), while in heterogeneous scenarios it is noticeably larger (14% versus 62.24%). This fact can be attributed to an increased number of balancing opportunities introduced by additional tasks.

Stability.

An important aspect of load balancing is the convergence toward a stable state where load is equally distributed among all the resources and no further rescheduling activity is performed. Because of the stochastic nature of some parts of the OZMOS algorithm, perfect stability cannot be possible. Moreover, a deterministic stable behavior might leave the system in an unbalanced state due to the presence of local minima. We determine the stability of our solution by analyzing the number of tasks that are migrated across the overlay during the simulation. This results also provides a rough figure of the network overhead, as migration of task descriptors accounts for the largest part of the traffic generated by the algorithm (the rest owing to Chord's signaling messages as well as *Notification* and *Relocation* agents). Figure 5 illustrates the amount of rescheduled tasks in homogeneous and heterogeneous overlays with different sizes. In homogeneous overlays the system is more unstable: this result can be attributed to the fact that the algorithm frequently tries to balance load differences equal to the size of each task, which leads to bouncing effects. On the contrary, in heterogeneous scenarios more diverse load balancing possibilities (with different task run times and node performance) lead to a more stable end result. These differences are also reflected in the variance measured across different simulation runs: for example, in overlays with 1024 nodes variances of 92% and 11% of the number of rescheduled tasks where observed after 10800 seconds in the homogeneous and heterogeneous scenarios respectively. Concerning the total number of tasks, Figure 6 shows that heterogeneous scenarios again perform better than their homogeneous counterparts. As expected, a larger amount of tasks results in more tasks being migrated in order to achieve system wide load balance.

From homogeneous to heterogeneous.

Figure 7 shows the convergence of the %RSD in different scenarios with varying degrees of heterogeneity. The algorithm is able to deal well with the differences in the run time performance of each node and of task run times, showing similar results as in the full homogeneous scenario. The influence of heterogeneous resource classes is evident when the corresponding scenario is compared to other results. The behavior of the algorithm in such different situations is reflected in the stability results depicted in Figure 8: heterogeneity leads to a faster convergence toward a stable state; however, the situation at the end of the simulation shows little differences in all scenarios but the ones with heterogeneous resource classes.

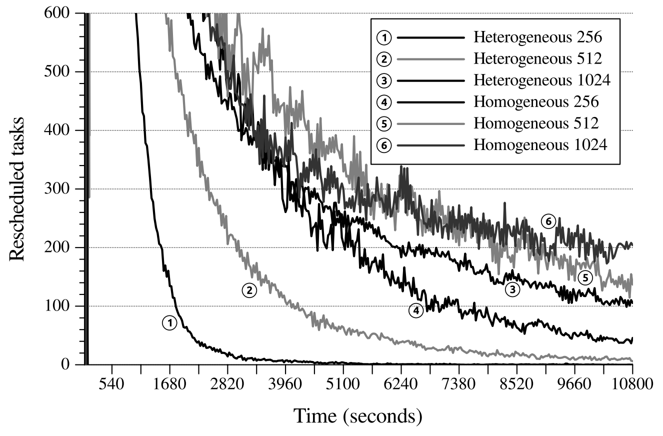

Figure 5: Stability (increasing overlay size)

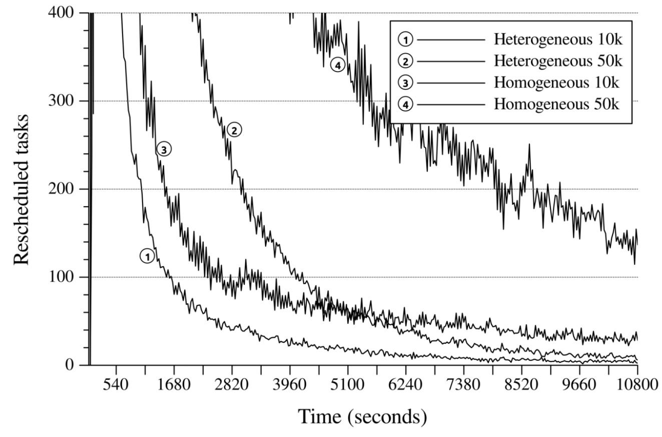

Figure 6: Stability (task count)

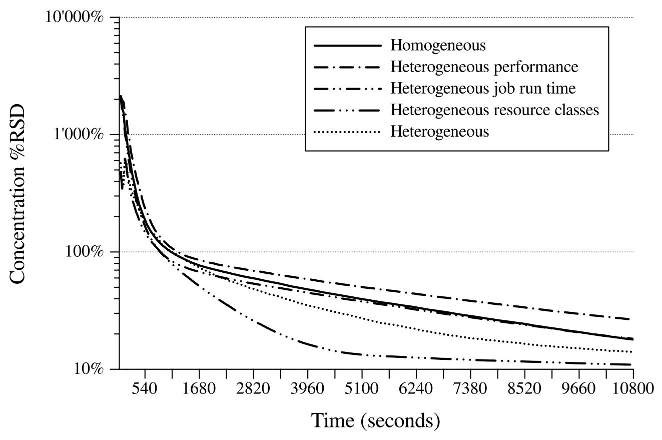

Figure 7: Load balancing performance (heterogeneity factors)

6. CONCLUSIONS

In this paper we presented a fully distributed algorithm and protocol for load balancing in homogeneous and heterogeneous grids based on the P2P paradigm. The proposed solution, called OZMOS follows the principle of osmosis to relocate tasks among adjacent nodes that are connected by means of a Chord structured overlay. To share information about the state of the grid as well as to relocate tasks, bio-inspired ant-like agents are employed. The key based routing capability of Chord is used to discover other nodes in the overlay, and to efficiently support relocation of incompatible tasks in heterogeneous grids. The proposed mechanism naturally supports heterogeneous resources, with the overlay organized such that resources with similar profiles are adjacent on the ring. The load balancing performance of the algorithm has been validated in different experimental scenarios, with both homogeneous and heterogeneous resources. The scalability of our solution concerning both the overlay size as well as the number of scheduled tasks was also confirmed. Future work will include a detailed sensitivity analysis of the parameters of the algorithm as well as an in depth experimentation with a more realistic grid setup which would include task execution and dynamic network conditions. The present design is also open to further extension, for example with the introduction of a locality-aware [29] overlay structure.

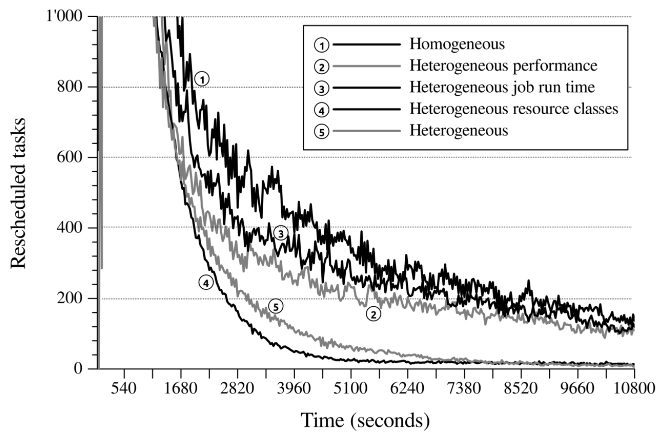

Figure 8: Stability (heterogeneity factors)

7. ACKNOWLEDGMENTS

This research has been carried out thanks to the financial support of the Swiss National Science Foundation, fellowship Nr. 134285.

8. REFERENCES

[1] Ian Foster and Adriana Iamnitchi. On death, taxes, and the convergence of peer-to-peer and grid computing. 2003.
[2] Jennifer M. Schopf. Ten actions when grid scheduling: the user as a grid scheduler. 2004.
[3] A. Andrzejak, A. Reinefeld, F. Schintke, T. Schütt, C. Mastroianni, P. Fragopoulou, D. Kondo, P. Malecot, G. Cosmin Silaghi, L. Moura Silva, P. Trunfio, D. Zeinalipour-Yazti, and E. Zimeo. Grid architectural issues: State-of-the-art and future trends. CoreGRID White Paper, May 2008.
[4] Ian Foster, Yong Zhao, Ioan Raicu, and Shiyong Lu. Cloud Computing and Grid Computing 360-Degree Compared. In *2008 Grid Computing Environments Workshop*. IEEE, November 2008.
[5] Ion Stoica, Robert Morris, David Karger, Frans M. Kaashoek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications.

In *SIGCOMM '01*, volume 31, New York, USA, October 2001. ACM Press.

[6] Ian Foster and Carl Kesselman. Globus: A metacomputing infrastructure toolkit. *The International Journal of Supercomputer Applications and High Performance Computing*, 11(2), 1997.

[7] Mathilde Romberg. The unicore architecture: seamless access to distributed resources. In *High Performance Distributed Computing*, 1999.

[8] K. Christodoulopoulos, V. Sourlas, I. Mpakolas, and E. Varvarigos. A comparison of centralized and distributed meta-scheduling architectures for computation and communication tasks in grid networks. *Comput. Commun.*, 32, May 2009.

[9] Janhavi Arvind Baikerikar, Sunil K. Surve, and Sapna U. Prabhu. Comparison of load balancing algorithms in a grid. *Data Storage and Data Engineering, International Conference on*, 0, 2010.

[10] K. Lu, R. Subrata, and A.Y. Zomaya. An efficient load balancing algorithm for heterogeneous grid systems considering desirability of grid sites. In *Performance, Computing, and Communications Conference, IPCCC 2006*, 2006.

[11] Amos Brocco, Apostolos Malatras, Ye Huang, and Beat Hirsbrunner. Aria: A protocol for dynamic fully distributed grid meta-scheduling. *ICDCS 2010*, 2010.

[12] Manish Arora, Sajal K. Das, and Rupak Biswas. A de-centralized scheduling and load balancing algorithm for heterogeneous grid environments. In *ICPPW '02: Proceedings of the 2002 International Conference on Parallel Processing Workshops*, Washington, DC, USA, 2002. IEEE Computer Society.

[13] Po-Jung Huang, You-Fu Yu, Kuan-Chou Lai, and Chao-Tung Yang. Distributed adaptive load balancing for p2p grid systems. In *Pervasive Systems, Algorithms, and Networks (ISPAN 2009)*, 2009.

[14] Ruchir Shah, Bhardwaj Veeravalli, and Manoj Misra. On the design of adaptive and decentralized load balancing algorithms with load estimation for computational grid environments. *IEEE Transactions on Parallel and Distributed Systems*, 18(12), 2007.

[15] Alexander Fölling, Christian Grimme, Joachim Lepping, and Alexander Papaspyrou. Decentralized grid scheduling with evolutionary fuzzy systems. In *Job Scheduling Strategies for Parallel Processing.* Springer Verlag, 2009. Lect. Notes Comput. Sci. vol. 5798.

[16] Rohit Gupta, Varun Sekhri, and Arun K. Somani. Compup2p: An architecture for internet computing using peer-to-peer networks. *IEEE Trans. Parallel Distrib. Syst.*, 17, November 2006.

[17] Arjav J. Chakravarti and Gerald Baumgartner. The organic grid: Self-organizing computation on a peer-to-peer network. In *ICAC '04: Proceedings of the First International Conference on Autonomic Computing*, Washington, DC, USA, 2004. IEEE Computer Society.

[18] Alberto Montresor, Hein Meling, and Özalp Babaoğlu. Messor: load-balancing through a swarm of autonomous agents. In *Proceedings of the 1st international conference on Agents and peer-to-peer computing*, AP2PC'02, Berlin, Heidelberg, 2003. Springer-Verlag.

[19] E. Bonabeau, M. Dorigo, and G. Theraulaz. *Swarm intelligence: from natural to artificial systems.* Oxford University Press, Inc., New York, NY, USA, 1999.

[20] Marco Dorigo and Thomas Stützle. *Ant Colony Optimization.* Bradford Company, Scituate, MA, USA, 2004.

[21] Junwei Cao. Self-organizing agents for grid load balancing. In *Proceedings of the 5th IEEE/ACM International Workshop on Grid Computing*, GRID '04, Washington, DC, USA, 2004. IEEE.

[22] Mohsen Amini Salehi and Hossain Deldari. Grid load balancing using an echo system of intelligent ants. In *Proceedings of PDCN'06*, Anaheim, CA, USA, 2006.

[23] Diego Castagna Francesco Palmieri. *Swarm-based Distributed Job Scheduling in Next-Generation Grids.* Springer, 2007.

[24] Azin Moallem and Simone A. Ludwig. Using artificial life techniques for distributed grid job scheduling. In *Proceedings of the 2009 ACM symposium on Applied Computing*, SAC '09, New York, NY, USA, 2009. ACM.

[25] Riccardo Poli, James Kennedy, and Tim Blackwell. Particle swarm optimization. *Swarm Intelligence*, 1(1), June 2007.

[26] Kay Dörnemann, Jörg Prenzer, and Bernd Freisleben. A peer-to-peer meta-scheduler for service-oriented grid environments. In *Proceedings of the first international conference on Networks for grid applications*, GridNets '07, Brussels, Belgium, 2007. ICST.

[27] Graham R. Nudd, Darren J. Kerbyson, Efstathios Papaefstathiou, Stewart C. Perry, John S. Harper, and Daniel V. Wilcox. Pace–a toolset for the performance prediction of parallel and distributed systems. *International Journal of High Performance Computing Applications*, 14(3), 2000.

[28] Ingmar Baumgart, Bernhard Heep, and Stephan Krause. OverSim: A flexible overlay network simulation framework. In *Proceedings of 10th IEEE GI/INFOCOM 2007, Anchorage, AK, USA*, May 2007.

[29] Weiyu Wu, Yang Chen, Xinyi Zhang, Xiaohui Shi, LinCong, Beixing Deng, and Xing Li. Ldht: Locality-aware distributed hash tables. In *Information Networking, 2008. ICOIN 2008. International Conference on*, 2008.