Natural Computing manuscript No.
(will be inserted by the editor)

The Grid, the Load and the Gradient

A bio-inspired approach to load balancing

Amos Brocco

Received: date / Accepted: date

Abstract An important concern for an efficient use of distributed computing is dealing with load balancing to ensure all available nodes and their shared resources are equally exploited. In large scale systems such as volunteer computing platforms and desktop grids, centralized solutions may introduce performance bottlenecks and single points of failure. Accordingly fully distributed alternatives have been considered, due to their inherent robustness and reliability. In extremely dynamic contexts, scheduling middlewares should adapt their job scheduling policies to the actual availability and overcome the volatility and heterogeneity typical of the underlying nodes. To deal with the dynamicity of a large pool of resources, self-organizing and adaptive solutions represent a promising research direction. Solutions based on bio-inspired methodologies are particularly suitable, as they inherently provide the desired features. In this paper we present a fully distributed load balancing mechanism, called OZMOS, which aims at increasing the efficiency of distributed computing systems through peer-to-peer interaction between nodes. The proposed algorithm is based on a CHORD overlay, and employs ant-like agents to spread information about the current load on each node, to reschedule tasks from overloaded systems to underloaded ones, and to relocate incompatible tasks on suitable resources in heterogeneous grids. By means of several evaluation scenarios we demonstrate the effectiveness of the proposed solution in achieving system-wide load balancing, both with homogeneous and heterogeneous resources. In particular we consider the load balancing performance of our approach, its scalability, as well as its communication efficiency.

Keywords Load balancing · Distributed Computing · Peer-to-Peer Networks

This research has been supported by the Swiss National Science Foundation
fellowship nr. 134285

Amos Brocco
Information Systems and Networking Institute, Department of Innovative Technologies
University of Applied Science of Southern Switzerland (SUPSI)
Via Cantonale, CH-6928 Manno, Switzerland
Tel.: +41-58-666 6589
E-mail: amos.brocco@supsi.ch

1 Introduction

The last decade has seen an unprecedented growth in the performance and number of systems connected to the Internet. The availability of high bandwidth connections has opened up new possibilities for leveraging the combined processing power and data storage capacity of millions of computers: distributed applications and solutions that once targeted high performance systems run by universities and research institutions can now count on the willingness of thousands or millions of users that share their own resources. This form of large scale distributed computing is commonly referred to as *peer-to-peer computing* [31] or *volunteer computing* [1,20]. These systems, which are also known as *desktop grids*, require little infrastructure and fewer investments than traditional grids, but trade their huge amount of available resources and raw processing power for a less dependable and highly volatile environment [35]: nodes can unexpectedly disconnect from the network, or reduce the amount of shared resources when the owner starts new local processes with higher priority. In contrast, traditional platforms such as *grids*, offer a higher quality of service, performance guarantees, and security in exchange for higher ownership, running, and management costs. Furthermore, whereas volunteer and peer-to-peer computing systems are highly heterogeneous and mainly targets embarrassingly parallel problems with few or no dependencies on the underlying platform, grids provide a greater level of configurability and homogeneity. In desktop grids, heterogeneity originates both from diverse runtime platforms and from variable communication facilities: each participating computer may not only share a different set of resources (hardware and software) but can also enjoy different connectivity with the rest of the system (bandwidth, latency). The complexity required to overcome these differences is exacerbated by the fact that applications often need to be tailored for the target platform (clusters, grids, clouds), thus an increase in the diversity of end systems may require an additional development effort. However, even though peer-to-peer and desktop grids cannot ensure the dependability, efficiency and security of centralized solutions, they still represent a suitable alternative to implement a computing infrastructure with smaller investments for a number of situations, e.g. on-demand video streaming [7] or analysis of large datasets [16]. Generally speaking, the choice of a computing platform comes down to the nature of the project: in this regard, traditional grid and desktop grid technologies should be considered as complementary, particularly in the light of a convergence between the two paradigms, as observed in [24]. Even in the realm of cloud computing, which is characterized by metered on-demand dynamic provisioning [30] supported by large datacenters, there have been proposals for fully distributed implementations [4,15]. As a consequence, it is still worth researching methods to exploit the benefits of decentralization and on solutions to increase its efficiency.

To maximize the throughput and overall efficiency of a large pool of computing resources it is necessary to by employ intelligent scheduling mechanisms. Allocation of tasks on the most appropriate resources is thus a prerogative to an optimal usage of distributed computing networks [40]. Whether the underlying platform is a traditional or a desktop grid, in order to make an effective use of distributed computing it is important to define scheduling policies that support load balancing and avoid overloading just a fraction of the available computers. In this sense, traditional grid systems have an advantage over their distributed coun-

terparts: schedulers are typically centralized, and enjoy full knowledge about the resources available within the virtual organization as well as their current status. User demands can thus be precisely matched to fulfill QoS agreements such as maximum response time, price, etc. Centralized solutions also simplify accounting and security, while allowing providers to enforce strict usage policies. However, centralized models also create single points of failure and incur in higher management costs in order to maintain a reliable infrastructure. For this reason, research has started to investigate solutions that strive to reduce deployment costs, increase reliability, and meet dynamic users' needs, envisioning flexible, autonomic, and self-manageable grids [2].

To tackle the problem of optimal scheduling, in this paper we present a fully distributed task load balancing mechanism for desktop grids that employs bio-inspired techniques to provide autonomous and self-organized operation. Our solution is based on the principle of osmosis, and employs ant inspired agents to reallocate tasks among the available resources. Osmosis is a self-regulatory process at the base of several natural processes, in particular of cellular exchanges. The algorithm, named OZMOS, is based on a variation of this process running on a CHORD [42] overlay, and achieves load balancing by relocating tasks between nodes both in homogeneous and in heterogeneous grid systems. This paper extends our previous work [10] with additional evaluations and a comparison with another fully distributed and bio-inspired load balancing mechanism [34]. 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. For simplicity, in the following of this paper we use the terms *task* and *job* interchangeably while referring to requests for resource allocations submitted to a distributed computing system.

2 Related work

Regardless of the computing infrastructure, be it either a traditional grid or a volunteer computing platform, resource management and scheduling remain important concerns to make efficient use of the available processing power. Even when dealing with very large pools of dispersed resources, as in the case of desktop grids, it is critical to assign jobs to the most appropriate nodes and avoid overloading just some of the systems. As our work is concerned with fully distributed solutions, in this section we put or focus on existing mechanisms for decentralized scheduling and load balancing, which mainly target grid computing. Scheduling in grids is managed by meta-schedulers, which assign jobs to available resources according to their existing local scheduling policies. An additional realm of our research is load balancing, which can be performed either statically or dynamically by the scheduler [5]: because in desktop grids resources and loads change with time and their exact characteristics are not known *a priori* [29], it makes sense to only consider the latter.

In traditional grid solutions, such as [23,38], centralized or hierarchical meta-schedulers are employed: these schedulers have access to a complete and up-to-date

knowledge about available resources, and can therefore perform optimal scheduling decisions. A form of hierarchical scheduling can be employed even in peer-to-peer scenarios: for example, in [28] an adaptive load balancing strategy for a super-peer based P2P grid is detailed. Super-peers are responsible for 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. Unfortunately, drawbacks arise from these designs: super-peers can create performance bottlenecks that hinder the scalability of the system and represent single points of failure that affect all managed peers and the robustness of the whole grid. In the light of these issues, research has proposed distributed approaches that strive to overcome the flaws of centralized solutions while retaining an acceptable quality of service with minimal communication overhead [14].

Examples of fully distributed scheduling with load balancing include [11], which introduces a protocol based on a cost-for-execution heuristic, [3], where a node retains jobs or reschedules them to its neighbors according to the actual load, and [3], where each node employs a heuristic based on local load to decide if a job is to be delegated to a neighbor. Another solution is presented in [41], and employs a decentralized and adaptive scheduling strategy with load balancing capabilities to equalize the expected execution time across nodes. Similarly, in [21], an adaptive decentralized algorithm based on evolutionary techniques is proposed. Finally in [26] 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. A comparable scheduling and load balancing mechanism is discussed in [19]: balancing is achieved through an economic model where agents attempt to achieve their goals and obtain the maximum profit from the rest of the system.

Distributed solutions exhibit noticeable complexity, because they need to minimize the costs of communication and synchronization while performing scheduling decisions in a dynamic environment using partial and possibly outdated information. Accordingly, self-organizing methodologies such as bio-inspired techniques are increasingly being considered. In this regard, some projects view the grid as a complex system where autonomic scheduling and load balancing solutions can be achieved by means of self-organizing agents executing on a peer-to-peer overlay. In [13], the scheduling process is modeled after a biological system: jobs are divided in subtasks that are assigned to an agent. Each agent is deployed on the grid in order to carry out the computation on suitable computational resources. Nodes can request work from other peers, effectively creating a task hierarchy in the network. Furthermore, each peer can adapt the amount of requested tasks based on past experience. A similar approach is presented in [18], with a meta-scheduling architecture based on ant-like agents that migrate on a DHT overlay: the structured overlay is used to maintain multicast communication trees and efficiently track jobs delegated to other nodes.

Another well known example of load balancing mechanism based on artificial life behaviors is Messor [34], which builds upon the swarm intelligence [8] and ant colony [17] paradigms. Messor nodes deploy ant-like agents in order to discover suitable resources and eventually trigger load balancing. If a node is overloaded, the goal of the deployed agent is to find an underloaded node; on the contrary, the goal of the agent is to look for overloaded systems. Agents wander on the

network and collect information about the load of visited nodes: this information is stored on each node, and can be exploited by other agents in order migrate toward nodes of major interest. After some time, and when a node with the desired load is found, the balancing takes place, and tasks are transferred from the most loaded discovered node to the least loaded one. Algorithms similar to Messor have been adopted by other platforms such as ARMS [12] or [39]. ARMS uses a swarm of ants that spread information about nodes connected to a hierarchical overlay. Each node is managed by a local scheduler that communicates with other peers in order to exchange tasks and balance the load. Another similar scheduling framework is proposed in [25], where ant-like agents are deployed upon submission of a task and wander on the network looking for suitable resources. The goals of each agent are to discover remote resources and to minimize both the overall makespan (the maximum time required to complete all jobs) and the response time. The ant paradigm is also employed in [32], by means of agents that wander on the network looking for resources that provide the best match for the requirements of a job. Ant agents can communicate through stigmergy, i.e. by leaving artificial pheromone trails indicating the fitness of each inspected resource. Other bio-inspired techniques, such as particle swarm optimization [37], can also be employed to achieve load balancing. For example, in [32] a distributed version of particle swarm optimization is used to evaluate the load in the vicinity of a node, and transfer tasks toward nodes with the highest fitness (i.e. nodes that have the lowest load and can thus contribute to a better balancing of the grid).

Our research aims at employing concepts of self-organization to solve the problem of efficient scheduling and load balancing in distributed computing. Accordingly, we propose a system based on bio-inspired agents wandering on a structured P2P overlay. To minimize the network overhead our solution relies on simple interactions between adjacent nodes and the structure of the P2P overlay, and does not employ an agent-based resource discovery mechanisms. Furthermore we make use of the the key-based routing capability of the underlying overlay to quickly move agents toward nodes that provide the resources needed for the execution of a job, and we exploit the overlay management protocol to organize and group nodes according to their capabilities. In contrast to other bio-inspired approaches, our solution efficiently supports both homogeneous and heterogeneous tasks and resources.

3 ozmos Protocol

The ozmos algorithm tackles the problem of efficiently relocating tasks among a large pool of distributed resources in a peer-to-peer overlay, achieving similar load on each node. Our solution is fully distributed, and employs two bio-inspired techniques, namely osmosis and ant-like mobile agents. Nodes manage a batch scheduling queue and can execute one or more tasks concurrently according to their performance characteristics. Each node can push some of its tasks to another node if it detects that the latter is less loaded. For this, we assume that tasks are independent, non-divisible, and non-preemptive: tasks cannot be relocated while running, and each tasks is considered as the smallest computational unit the execution which does not depend on any other unit. An important aspect of our solution is support for both homogeneous and heterogeneous distributed

computing: in a homogeneous scenario all tasks have the same requirements, and each node is expected to be able to execute any task (although with different performance characteristics). Conversely, in a heterogeneous grid each task has different requirements (such as CPU architecture, operating system, libraries, etc.) that can be provided only by a subset of the nodes. In the following of this section we provide a detailed discussion of OZMOS and its load balancing process.

3.1 Osmosis

Osmosis [27] is a natural phenomenon that occurs when two solutions at different concentrations are separated by a semi-permeable membrane that allows only the molecules of the solvent (for example, water) to pass through. In this situation the solvent moves from the solution with the lowest solute concentration toward the other: when the concentration on both sides of the membrane is equalized it remains stable, although the solvent continues to flow in equal amounts across the membrane. When the system is in an unbalanced state, the force required to prevent the solvent from moving, the osmotic pressure, depends on the difference between the concentrations in the two solutions (also called the osmotic gradient). Figure 1 depicts an example of the process: the concentration of the solute on the left side of the bin is initially at 83%, whereas on the right side it is at 25%. The solvent molecules pass through the membrane separating the solutions, and both sides of the bin finally reach an equal concentration of 50%. The process stabilizes because the osmotic pressure generated by the increased amount of solvent on one side of the membrane prevents further molecules from passing through. Osmosis is advantageous for cellular life, as it enables water to move in and out of a cell without requiring energy (passive transport). Our load balancing solution imitates this mechanism by performing only local task exchanges (i.e. between neighboring nodes) from nodes with higher loads, toward nodes with lower load. A benefit of local interaction between nodes is a lower communication overhead, which allows nodes to keep more up-to-date information about peers in their vicinity while generating minimal traffic.

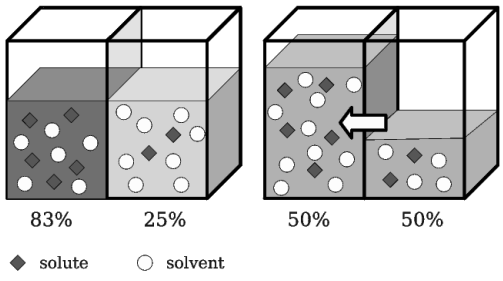

Fig. 1: Osmosis

3.2 Chord

OZMOS requires an underlying peer-to-peer overlay where nodes can exchange information about their status and mobile agents can migrate to relocate tasks. For our purposes we chose CHORD [42], which maintains a P2P overlay and implements a distributed hashtable (DHT). The overlay is structured as a ring, and each node is assigned a unique identifier of $b = 160$ bits. Within the ring, nodes are ordered according to their identifier (forming a circular identifiers' space), and maintain pointers to their successors and to their predecessor in the overlay. As a distributed hashtable, CHORD maps keys to values that can be stored on nodes of the overlay. Each value is assigned a key in the same domain as nodes' identifiers: key-value pairs are published on the node whose identifier is the closest to key of the data. To look up for a key a query message is forwarded in the overlay until the corresponding node is found. To speed up the look up process, each node maintains a finger table with shortcuts to other nodes at increasing distances in the overlay. CHORD enables very efficient information retrieval, with its ability to route a query to destination in $O(logN)$ hops in an overlay composed of N nodes. Because the only requirements for the underlying P2P overlay are an ordered ring topology and a routing mechanism, it is possible to replace CHORD with SELF-CHORD [22] in order to implement a complete resource discovery and scheduling solution based on self-organized methodologies.

3.3 Local and remote concentrations

To replicate the osmosis process for load balancing it is necessary to define what constitutes the solvent, respectively the solute, and the concentration of the solution. In our scenario, the solute component are the computational resources on each node, and its amount is a value inversely proportional to the computing performance of a node. As with real osmosis, our solute cannot be transferred across the network. Conversely, the solvent molecules are the tasks scheduled on a node, which can be distributed among available resources so that each system is equally loaded. Accordingly, we define the local *concentration* of a node as the time required to process all tasks in its queue. This definition enables us to consider a network of connected resources with different performance characteristics: given the same amount of workload, the concentration value is higher on nodes with lower performance. More precisely, 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. The speed index is a floating point value that expresses the performance of the node compared to a baseline system, which has $speed_M = 1$. This baseline system is also used to determine j_{ert} (for example by means of a profiling tool such as PACE [36]); generally we assume that a node with $speed_M = t$ will complete jobs

t times faster than the baseline system. Furthermore we 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 periodically determine their load balancing actions by comparing the local concentration with that of neighboring nodes in the network. 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 load balancing state in the vicinity of the node, whereas the probe offers an overview of distant areas of the overlay. Because the p and s links match the underlying ring structure, they typically do not change during the life of the node. On the contrary a new probe address is periodically received from a different peer and the link is thus more volatile.

3.4 Resource identifiers

An important feature of OZMOS is its support for both homogeneous and heterogeneous grids. In a homogeneous scenario, each node is assigned a random unique identifier of m bits (typically $m = 160$) that matches the one used by CHORD. To deal with heterogeneous scenarios, we restrict ourselves to a finite number of classes $s \geq 0$ instead of dealing with arbitrarily heterogeneous resources. Resource classes can be assigned according to the architecture of the machine, and its operating systems: a task can request a specific resource class and thus set the requirements for its execution. In order to determine the resource class of each node, its CHORD identifier is generated so that it contains both the class identifier as well as a randomized key. More precisely, 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 2).

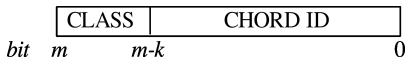

Fig. 2: Node identifier with class field

This mechanism allows each node to determine the class of known nodes (successor, predecessor, and finger nodes), without explicitly querying them. Since the value $k \ll m$ this modification does not affect the operation of the overlay management protocol. On the contrary, since CHORD maintains a global ordering of the nodes according to their identifier, all peers belonging to the same class will connect to adjacent positions in the ring. In this regard, OZMOS can seamlessly support heterogeneous grids, which can be viewed as a series of connected homogeneous systems. For simplicity, tasks are similarly assigned unique identifiers whose prefix also corresponds to a resource class: by doing so, the task identifier enables its unambiguous tracking and implicitly declares the class of resources required for its execution. In the present implementation of OZMOS we have not considered the case of overlapping classes. Nonetheless, minor adjustments to the algorithm can

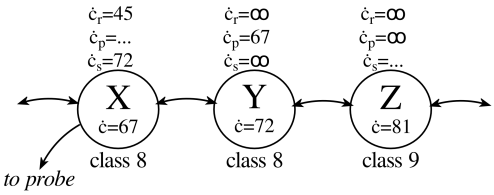

Fig. 3: Example of heterogeneous grid (performance indices omitted)

be introduced to allow load balancing between different but overlapping classes if those map on adjacent regions in the ring.

3.5 Agents

To exchange information between nodes and to relocate tasks across the overlay we employ ant-like mobile agents. Ants are software agents with strong and transparent migration capabilities: agents can decide to move to another node, and have their runtime state fully transferred to the specified location in the network, where execution is resumed. These agents follow the paradigms of swarm intelligence [8] and ant colonies [17]: the operation of the system is distributed across a large set of ant-like agents that have only a limited local knowledge of the system. Each ant behaves in an autonomous way, and its solving capabilities are limited to simple tasks; nonetheless, the emerging collaborative behavior of the whole colony can be used to solve complex goals without centralized control. Swarm systems are typically also self-organizing, and can adapt to changes in the environment (for example, the topology of the network). Furthermore the resulting system is generally more robust toward failures, and can tolerate the death of some individuals of the colony.

OZMOS employs three types of agents with different behaviors; while executing on a node, each agent has 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. It is important to note that we don't currently address security issues such as validation of incoming agents before their execution. We are aware that solutions that can be deployed into real-world grids must take care of the problem, and should not allow execution of arbitrary software or processing of arbitrary data without strict control. A solution that can be easily implemented is verifying the origin of agents and tasks and discard data coming from untrusted sources. Because our main concern is load balancing, the protocol does not currently support recovery from node or network failures, and both queued and executing tasks are lost if a node fails. Finally, for the actual job migration we assume that an underlying data transfer mechanism which takes care of relocating both the task application as well as its data is available.

Notification Nodes periodically send a *Notification* agent to their predecessors, successors, and to randomly chosen nodes. The task of *Notification* agents is to inform both ring neighbors and a remote node about the local concentration and performance characteristics. Whereas the predecessor and successor addresses are

clearly defined by the structure of the overlay, the random node's address has to be drawn from the local finger table and the successors' list. To support load balancing in heterogeneous grids, agents are not sent to nodes whose resource class is different from that of the origin node. Hence the value of remote concentrations of *incompatible* nodes remains equal to its initial value of ∞. A suitable random node of the same resource class can be retrieved from the finger table by checking its identifier: if the node cannot retrieve another peer with an admissible identifier the notification phase is simply skipped. 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 stores the local normalized concentration and performance index values in its memory 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 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 (the *probe*) that represents an additional direction for off-loading tasks to. Figure 3 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 because their class is different, and 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 $\dot{c}_r$, which can be used for load balancing.

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: $\quad \dot{c}_s := c$
7: $\quad perf_s := v$
8: **else**
9: $\quad$ **if** $x = p$ **then**
10: $\quad\quad \dot{c}_p := c$
11: $\quad\quad perf_p := v$
12: $\quad$ **else**
13: $\quad\quad r := c$
14: $\quad\quad \dot{c}_r := c$
15: $\quad\quad perf_r := v$
16: $\quad$ **end if**
17: **end if**

Osmosis *Osmosis* agents are used to relocate tasks from a node with high concentration toward a node with low concentration. Each node M periodically determines the osmotic gradient toward other nodes, by computing the difference between the local concentration and that of each node $n \in \{p, s, r\}$, namely

$c_M - \frac{\dot{c}_n}{perf_n}$. Because nodes can only decide to offload some of their tasks and cannot request them, only the node x corresponding to the highest positive difference is chosen as candidate for the load balancing process. In order to equalize local concentrations on both scheduling queues, 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 follows:

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

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, starting with those with the shortest estimated run time. 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. This threshold is necessary because the sum of task run times can only approximate the requested value. The migration process is carried out by *Osmosis* agents, whose behavior is detailed in Algorithm 2. Agents are given a list of task descriptors and a direction to follow in the ring. When an agent arrives on its target node the local concentration is evaluated: if this value is lower than that of the succeeding one (according to the actual traveling direction), the tasks carried by the agent are released 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 found, 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.

Relocation In a heterogeneous grid, tasks might be submitted to a node whose class is different and the resources do not permit 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 of incompatible jobs by migrating them into another scheduling queue. In contrast to the *Osmosis* agent, the *Relocation* one does not use the predecessor and successor links to move on the overlay, but employs CHORD's key based routing to directly jump to a node whose class is compatible with the tasks being relocated.

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**

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 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**

4 Evaluation

Through a series of experiments in both homogeneous and heterogeneous scenarios, we aim at validating the load balancing performance of ozmos as well as its communication efficiency. In particular, to determine if the load is well balanced across the grid we measure the relative standard deviation of the load across all nodes. This measure considers both the total expected running time of each scheduling queue as well as the performance of each node. Furthermore we consider the traffic generated by the agents created by the algorithm. Because we employ simulated workloads, the transfer of the actual data tied with each job is not considered in our analysis. All experiments are conducted using the OverSwarm [9] platform, based on OverSim [6] and the OMNET++ [43] discrete event simulator, which

reproduces latency and takes into account the delays and queuing effects of an underlay network. This section presents and discusses the details of the considered evaluation scenarios.

General setup In order to obtain statistically valid data, each experiment consists of 5 simulation runs using the same parameters, with each run covering about 5 hours and 30 minutes of activity. Unless otherwise stated, each simulation is conducted using an overlay of 2048 nodes, managed by the CHORD protocol. At the beginning of each experiment, nodes join the overlay with an average frequency of 1 node every second. The duration of this initialization phase thus varies depending on the final size of the overlay (from 1024 seconds to 4096 seconds). One of the nodes receives all the work load, namely a set of 10000 to 100000 tasks. The main parameters of the algorithm are kept constant throughout all experiments: the osmosis threshold ϵ is set to 1.05, which means that a node not may not offload more than 105% of the required amount while load balancing. To inform neighbors about the local concentration level, every 30 seconds every node deploys its *Notification* agents. Local and remote concentrations are compared every 60 seconds, eventually triggering the creation of *Osmosis* agents to perform load balancing. Each *Osmosis* agent can travel at most 10 hops in the overlay following a path toward a lower concentration: after 10 hops, carried tasks are released and scheduled on the current node. In heterogeneous scenarios incompatible tasks, if present in the scheduling queue, are relocated to the appropriate node every 120 seconds, with only one class of incompatible jobs relocated at a time. All communication between nodes is performed asynchronously and assuming a reliable network, where neither communication nor node failures occur.

Homogeneous scenarios In homogeneous scenarios all nodes are grouped into a single resource class, and can execute the same tasks although with different performances. At the start of each experiment, one node is given all the 50000 tasks, with a run time j_{ert} chosen uniformly at random in the continuous interval between 15 and 45 minutes. 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. To determine the scalability of our solution, experiments with varying overlay sizes and task count are also conducted: more specifically, we consider additional simulations with 1024 and 4096 nodes, as well as with 10000 and 100000 submitted tasks.

Heterogeneous scenarios In heterogeneous scenarios both nodes and tasks are attributed a random resource profile out of the 64 available classes. Computing power of each node is randomly assigned as in homogeneous scenarios, and at the start of each simulation 50000 jobs are submitted to a random node. On each node, we distinguish between compatible and incompatible jobs: the former require a set of resources of the same class as the current node, whereas the latter require a different class and need to be rescheduled on appropriate nodes by means of *Relocation* agents. As in homogeneous scenarios, we evaluate the scalability of the algorithm by performing tests with overlays of different sizes (1024, 2048 and 4096 peers) and with a varying number of tasks (10000, 50000, and 100000). Additional network scalability experiments where the proportion between the number of nodes and the number of classes is kept constant are also

considered, namely by defining 32 classes in networks of 1024 nodes, and 128 classes with 4096 nodes. Finally, we conduct experiments with 2048 nodes and a different number of classes (32 and 128). In all heterogeneous scenarios the number k of high-order bits of the node's identifier used for storing the resource class is computed as $k = log_2(\text{number of classes})$.

Scenarios without probe nodes To determine the importance of *probe* nodes in the load balancing process we perform a series of experiments, both in homogeneous and heterogeneous conditions, without these additional links.

Comparison with a Messor-like algorithm In order to better understand how OZMOS compares to another fully distributed load balancing solution, experiments with a protocol inspired by MESSOR [34] are also performed. Nodes periodically deploy ant-like agents to discover overloaded or underloaded nodes and to obtain an overview of the status of the network. An agent can behave in two different ways: *search max* and *search min*. While in the *search max* state, the agent wanders in the network looking for nodes with excessive load (i.e. greater than the average load observed in the grid): the address and load of each visited node are stored in a list carried by the agent. When an overloaded node is found the ant-like agent might decide to switch to the *search min* and look for underloaded node (i.e. with a load lower than the average): while in this state, the ant wanders until it finds a suitable node. Subsequently the ant requests a job transfer from the overloaded node to the underloaded one, and switches back to the *search max* behavior. On each visited node, agents store information about the load of previously seen peers to allow nodes determine the average load on the grid and to help subsequent ants find better paths in the overlay. On each node, an agent is instanced every 480 seconds, and can travel up to 100 steps in the overlay. The behavior of each agent is determined by parameters and thresholds: in our simulations we defined these values according to [34] and an implementation of the protocol available in [33]. The exploration probability is set to 0.2 while looking for underloaded nodes (*search min* state) and 0.95 while searching for overloaded nodes (*search max* state). At each step, if the current node satisfies the criteria of the current state, an agent has a probability of 0.2 of continuing with its wandering instead of switching its state. The switch between the two states is determined by random probability of 50%, and can occur if the relative difference between the most loaded node and the least loaded node is less than 0.01. An agent can carry information about at most the last 16 visited nodes. Similarly, the maximum size of the local view stored on each node is also 16 entries: when there's no space left, a least-recently-updated strategy is used to remove the oldest entries.

Measurements Measurements of the load balancing process start after the initialization phase; accordingly the reported results appear shifted on the time axis depending on the size of the overlay. In particular, in overlays of 1024 nodes the load balancing process and our measurements start after about 1024 seconds, whereas with 2048 and 4096 nodes load balancing and measurements begin after 2048 and 4096 seconds respectively. 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

the average RSD over all classes is considered. Regarding the stability of the algorithm, the average number of tasks that are rescheduled in the previous 300 seconds is examined: this measurement is reported as a percentage of the overall number of tasks, and can exceed 100% if the same tasks are rescheduled multiple times during the considered period. To determine the scalability of the algorithm we compare the results obtained both in overlays of different sizes, as well with different workloads.

5 Results

In this section we report and analyze the results of the previously presented simulation scenarios. We first describe the experimental results concerning the load balancing capabilities of the algorithm; subsequently, we focus on the scalability of the algorithm in relation to the size of the network. Next, our discussion concentrates on to the stability of the load balancing process and its convergence toward a state of equilibrium. Subsequent experiments determine the response of the system to different loads (number of scheduled tasks) and the importance of probe links, as well as the influence of the number of classes in heterogeneous scenarios. Finally, we compare the results obtained with OZMOS with those of the MESSOR-like algorithm. Each evaluation scenario is labeled with a letter, according to Table 1. For each scenario of experiments we also present data concerning the generated traffic (average over 5 runs, as well as standard deviation), in order to give an insight of the overall efficiency of our approach.

Load balancing performance and scalability The first set of results concerns the load balancing performance of OZMOS on both homogeneous and heterogeneous grids when dealing with 50000 tasks. We consider three overlays of different sizes: 1024 (scenario A), 2048 (B), and 4096 (C) nodes. As shown in Figure 4, in homogeneous scenarios our algorithm is able to quickly balance the load across nodes after the start of the process. Unsurprisingly, by comparing the results obtained with 1024 and with 4096 nodes it is possible to observe a slight decrease of the the convergence rate as the network grows bigger. On the contrary, heterogeneous scenarios (Figure 5, scenarios D, E, and F) exhibit a greater difference between the different scales. This result is due to the different number of nodes belonging to each of the 64 classes: from an average of 16 nodes per class in an overlay of 1024 nodes, up to 64 in overlays of 4096 nodes. It is interesting to notice how the standard deviation in homogeneous overlays is sensibly lower than in heterogeneous ones, meaning that a lower number of tasks negatively affects the load balancing performance. Although not shown in the graph, all incompatible tasks are quickly relocated to suitable nodes in the early stages of the simulation.

	#Nodes	#Tasks	Resources	#Classes	Bytes/s per node	StDev (5 runs)
			OZMOS			
A	1024	50k	Homogeneous	-	115.2	0.22
B	2048	50k	Homogeneous	-	124.13	0.18
C	4096	50k	Homogeneous	-	134.22	0.12
D	1024	50k	Heterogeneous	64	107.89	0.30
E	2048	50k	Heterogeneous	64	118.84	0.126
F	4096	50k	Heterogeneous	64	129.24	0.06
G	2048	10k	Homogeneous	-	122.83	0.13
H	2048	100k	Homogeneous	-	125.42	0.14
I	2048	10k	Heterogeneous	64	117.28	0.05
J	2048	100k	Heterogeneous	64	120.29	0.05
K	2048	50k	Homogeneous.w/o probe	-	117.04	0.21
L	2048	50k	Heterogeneous.w/o probe	64	114.95	0.10
M	2048	50k	Heterogeneous	32	120.20	0.09
N	2048	50k	Heterogeneous	128	116.99	0.09
O	1024	50k	Heterogeneous	32	110.50	0.04
P	4096	50k	Heterogeneous	128	128.52	0.09
			MESSOR			
Q	1024	50k	Homogeneous	-	1187.05	2.42
R	2048	50k	Homogeneous	-	1082.28	3.25
S	4096	50k	Homogeneous	-	739.16	75.55
T	2048	10k	Homogeneous	-	1077.53	4.72
U	2048	100k	Homogeneous	-	1080.84	2.35

Table 1: Evaluation scenarios and generated network traffic (including CHORD protocol)

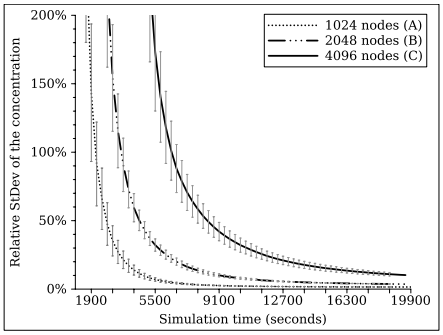

Fig. 4: Load balancing, homogeneous resources, different overlay size

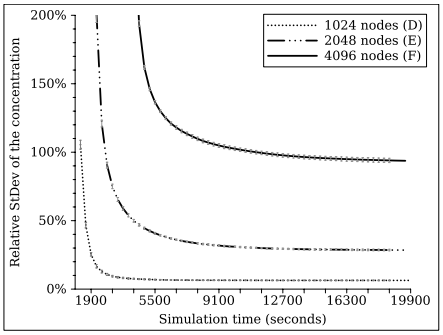

Fig. 5: Load balancing, 64 heterogeneous resources classes, different overlay size

Stability The stability of the load balancing process is determined by the number of tasks rescheduled by the algorithm over a period of time. As the load of each node equalizes, the number of tasks migrated by the algorithm should reduce. In contrast to real osmosis, where solvent molecules continue to flow even after equal concentrations are obtained, in a distributed environment we aim at limiting bandwidth consumption by preventing tasks from being migrated if that does not further improve the load balancing state of the system. Unfortunately, because

no global knowledge is available, small errors can still trigger minor rescheduling activities.

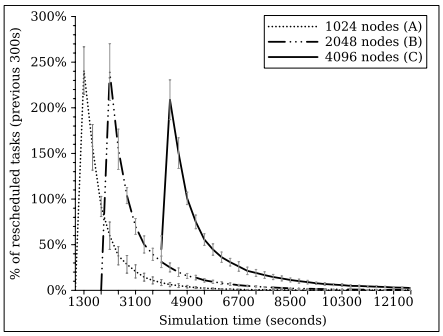

Fig. 6: Stability, homogeneous resources, different overlay size)

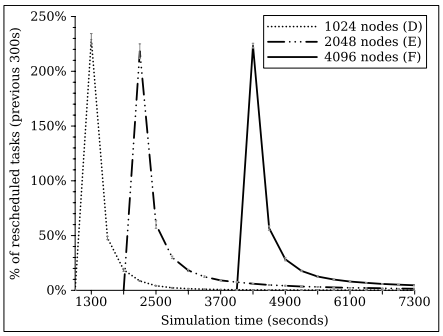

Fig. 7: Stability, 64 heterogeneous resources classes, different overlay size

As shown in Figures 6 and 7 most of the load balancing activity takes place as soon as the process starts. The number of rescheduled tasks quickly decreases and only very few tasks are subsequently migrated. These results demonstrate that the load balancing process converges toward a stable state.

Scalability (tasks) Another important evaluation concerns the balancing performance with varying load. Hence, we consider three scenarios with an increasing number of tasks (10000, 50000 and 50000) scheduled on a network of 2048 nodes. Figure 8 shows the results in a homogeneous system, whereas Figure 9 reports data concerning a heterogeneous grid. Surprisingly, in both homogeneous and heterogeneous scenarios a lower number of tasks negatively affects the load balancing process. The reason for this behavior is that a smaller number of tasks translates into fewer possible combinations for optimal scheduling on the grid. In contrast, with a sufficient number of tasks and thus a greater variety of run time lengths, an optimal solution can be easier to obtain.

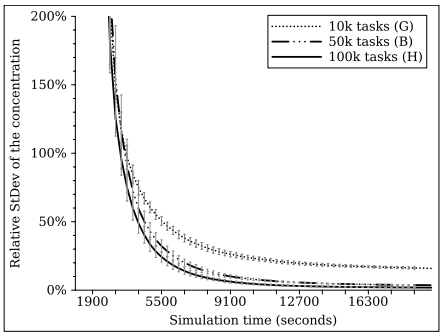

Fig. 8: Load balancing, homogeneous resources, different number of tasks

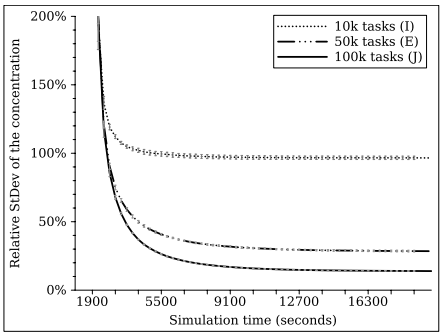

Fig. 9: Load balancing, 64 heterogeneous resources classes, different number of tasks

Stability (tasks) With scenarios G, B, and H (Figure 10) and I, E, J (Figure 11) we determine the stability of the algorithm in homogeneous and heterogeneous situations respectively. All experiments are conducted in overlays of 2048 nodes. In simulations with heterogeneous resources tasks are randomly assigned one of the 64 possible classes. No significant difference can be observed in the results, reflecting the ability of the algorithm to support increasing amount of tasks without producing unstable behaviors.

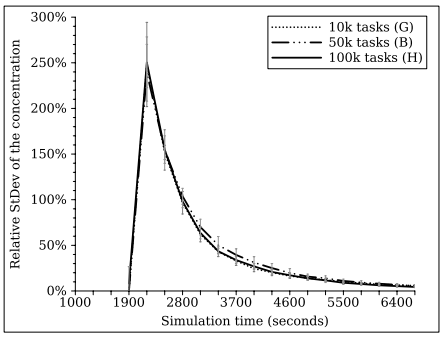

Fig. 10: Stability, homogeneous resources, different number of tasks

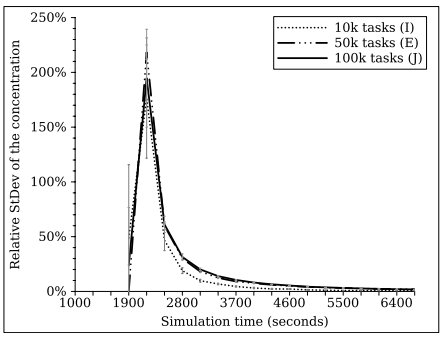

Fig. 11: Stability, 64 heterogeneous resources classes, different number of tasks

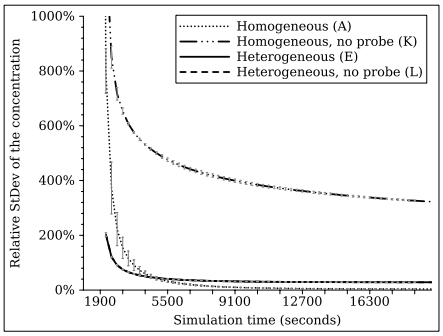

Fig. 12: Load balancing, overlay of 2048 nodes, with and without probe

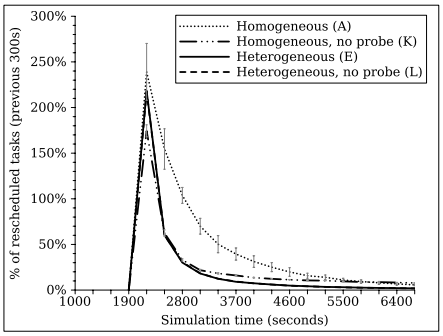

Fig. 13: Stability, overlay of 2048 nodes, with and without probe

Importance of the probe As discussed in Section 3 an important activity carried out by *Notification* agents is informing a *random* node beside predecessors and successors on the ring about the current load. The nodes from which the agent originates are considered as *probes* by the receiving peers. Because information about probes provide an insight of the load balancing status in remote part of the overlay, better and faster rescheduling decisions can be made. In particular, probe links prevent situations where the grid is in a globally unbalanced state because the difference between the concentration of adjacent nodes on the ring is too small to trigger a load balancing response. Figures 12 and 13 show the results obtained with and without probe nodes, concerning the load balancing perfor-

mance and the stability of the algorithm respectively. In homogeneous scenarios the lack of probe nodes considerably worsen the effectiveness of the load balancing process, with a standard deviation of the concentration that remains higher than 300%. On the contrary, in the considered heterogeneous system (where the 2048 nodes are divided into 64 classes) the absence of probe nodes does not influence the effectiveness of the algorithm, because the number of nodes in each class is very small. However, in larger scale system probe links would be required even in heterogeneous scenarios to ensure satisfying performance.

Varying the number of nodes-per-class As shown in the previously presented experiments, results obtained in heterogeneous scenarios depend on the number of nodes in each resource class. In order to understand how this number affects the performance of OZMOS, we analyze the data gathered through simulations with different nodes-per-class proportions. In this regard we first evaluate heterogeneous systems with different sizes (1024, 2048, and 4096) and a correspondingly different number of classes (32, 64, and 128 respectively) that maintain a constant 32 nodes-per-class proportion; second, we considered a system with 2048 nodes and a varying number of resource classes, namely 32, 64, and 128.

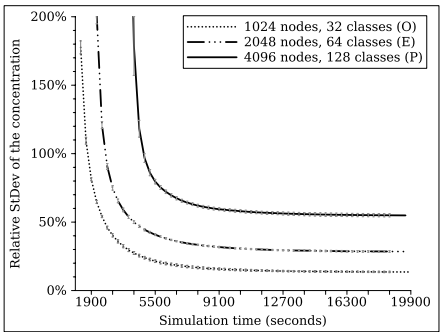

Fig. 14: Load balancing, heterogeneous overlays, same nodes-classes ratio

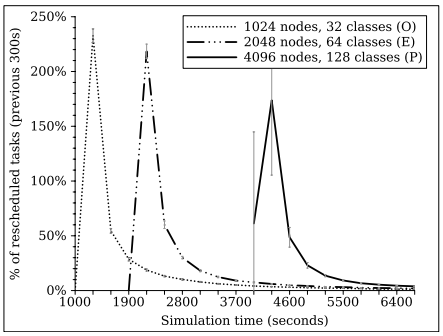

Fig. 15: Stability, heterogeneous overlays, same nodes-classes ratio

Figures 14 and 15 illustrate the behavior with a constant ratio. The convergence of the standard deviation of the concentration is similar at all scales, but reaches different lower bounds. In general, the standard deviation seems to worsen as the number of nodes and number of classes proportionally increases. To better understand the relation between the number of tasks, the number of nodes, and the number of classes, additional experiments are necessary. In this regard, Figures 16 and 17 report the results obtained through simulations on an overlay of constant size (2048 nodes), where the number of classes varies from 32 to 128.

Surprisingly, in these experiments a higher number of classes (128) provides the best results, even though each class is only represented by 16 nodes. This phenomenon clashes with our previous hypothesis, and can only be traced back to a strong dependency between the three aforementioned factors. By keeping the number of tasks constant, an increase in the number of classes reduces the total amount of jobs in each class and worsens the performance of the load balancing algorithm, as shown by previous experiments. However, a higher number of

classes also reduces the number of nodes belonging to each class, meaning that the load balancing process operates on a smaller system and can thus be more effective. Hence the number of classes to be used in the system should be carefully determined and should take into account the projected scale of the network.

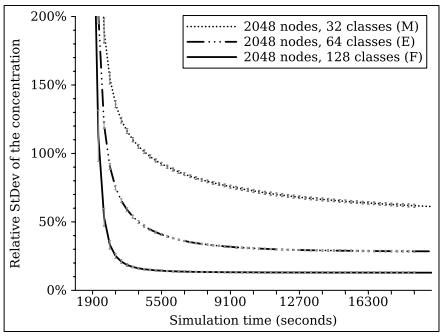

Fig. 16: Load balancing, 2048 nodes heterogeneous overlays, different nodes-classes ratio

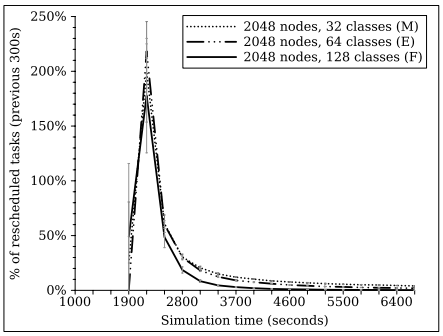

Fig. 17: Stability, 2048 nodes heterogeneous overlays, different nodes-classes ratio

Scalability comparison against Messor The last set of results concerns a comparison between OZMOS and a MESSOR-like algorithm. Results obtained with the latter are important because they provide a baseline to understand the advantages and drawbacks of our approach. This is a key factor, since both solutions are based on fully distributed bio-inspired solutions and aim for the same goals of self-organization and adaptivity. All experiments conducted with MESSOR are run on homogeneous grids: all nodes are able to execute all tasks, but with different performance characteristics (i.e. processing power and number of parallel tasks).

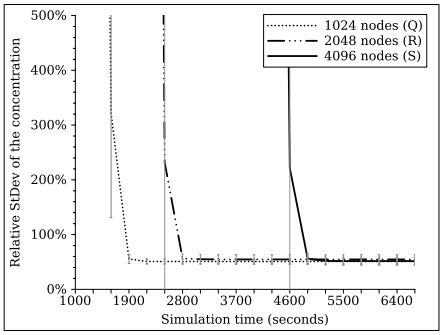

Fig. 18: MESSOR, load balancing

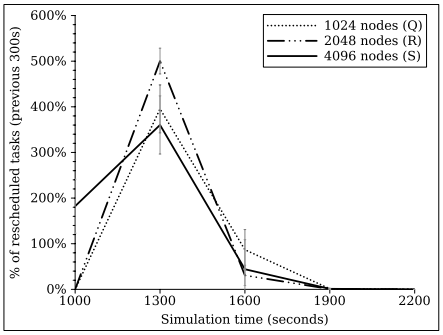

Fig. 19: MESSOR, stability

Figures 18 and 19 illustrate the convergence of the load balancing and the stability of the process respectively. It is clear that MESSOR is able to balance the whole system very quickly, and in contrast to OZMOS it does not exhibit noticeable variations when the size of the overlay is increased. By employing several agents that wander on the network collecting information about the load of each visited

node, MESSOR nodes can discover even distant regions of the network. The stability of the system after a balanced state has been reached is also highlighted in Figures 20 and 21: even with a large number of tasks the algorithm stops most of the rescheduling activities after 1900 seconds, with a relative standard deviation of the concentration leveling off at 50%. At this point, dismissing the fact that MESSOR does not support heterogeneous networks, a comparison between this algorithm and OZMOS seems to favor the former. To get a clearer picture, in the following section network another element of comparison will be taken into account: communication efficiency.

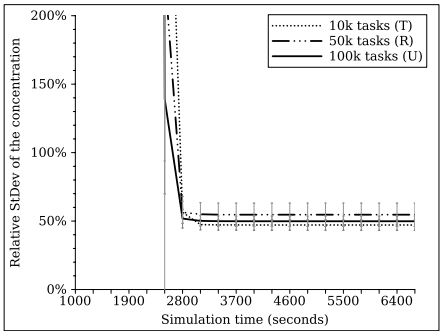

Fig. 20: MESSOR, load balancing

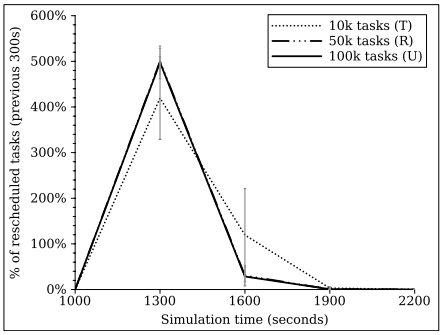

Fig. 21: MESSOR, stability

Network overhead To measure the bandwidth consumed by the algorithms in each scenario we measured the average amount of data sent by each node per second. Beside signaling messages originating both from the overlay management and the load balancing application, we evaluate the cost of transferring tasks between nodes. For this only the size of task descriptors is considered: each descriptor contains a unique identifier for the job, as well as a list of required resources. As shown in Table 1, both in homogeneous and in heterogeneous grids the overall bandwidth consumed by the algorithm remains minimal. Figure 22 illustrates the distribution of the traffic between CHORD and the load balancing application. Heterogeneous scenarios require additional bandwidth due to the use of key-based routing to reschedule incompatible tasks. Increasing either the network size or the number of tasks has minimal impact on the traffic generated by each node, which proves that our approach is indeed scalable.

The traffic generated by OZMOS is about ten times lower than with MESSOR: our solution is thus able to operate more efficiently while retaining satisfactory performance. The traffic generated by MESSOR is required to keep up-to-date information about the average load in the grid in order to respond to changes. Unexpectedly, with MESSOR an increase in the size of the grid has a noticeable benefit on the overall bandwidth consumption. This phenomenon is linked to the fact that with less nodes the load balancing requires more time and more rescheduling operations to reach stability. Moreover, because the number of available resources is lower, each agent must wander for a higher number of steps in order to find suitable nodes.

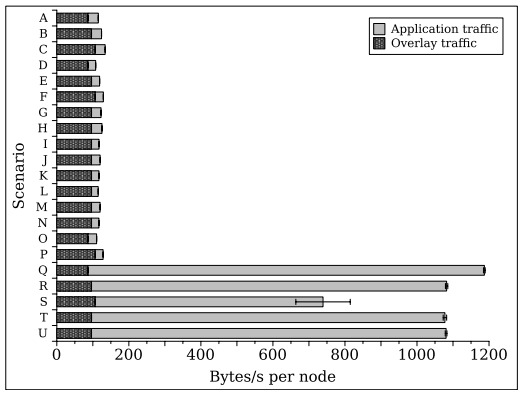

Fig. 22: Distribution of traffic between overlay management and load balancing application

6 Conclusions

This paper proposed a fully distributed algorithm for dynamic load balancing in homogeneous and heterogeneous computing systems such as desktop grids. Our solution is completely decentralized, and is based on P2P interaction between each computational node. The load balancing mechanism, called OZMOS, is inspired by osmosis, a physical process upon which many fundamental biological phenomenons are based. The proposed algorithm relocates tasks among adjacent nodes that are connected by means of a CHORD peer-to-peer overlay; ant-like mobile agents are used to share information about the status of the system and to balance loads between the available resources to improve the efficiency of the entire grid. The proposed mechanism easily supports grids with heterogeneous resources, by exploiting the overlay management algorithm and group resources with similar profiles together. The key based routing capability of CHORD is used to discover other nodes in the overlay, and to support relocation of incompatible tasks toward nodes with suitable resources. By means of several experimental scenarios, with both homogeneous and heterogeneous resources, the load balancing effectiveness of our approach has been verified. Furthermore, simulations with overlays of varying sizes as well as with different workloads proved that our solution is scalable. Finally, a comparison with another fully distributed approach demonstrated that OZMOS can achieve satisfying results while consuming considerably less bandwidth. Future work will focus on a more complete definition of concentration values, for example to take into account the cost of job transfer. Moreover, the introduction of a locality-aware [44] overlay structure will also be investigated.

7 Acknowledgments

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

References

1. Anderson, D.P.: Boinc: A system for public-resource computing and storage. In: Proceedings of the 5th IEEE/ACM International Workshop on Grid Computing, pp. 4–10. IEEE Computer Society, Washington, DC, USA (2004)
2. Andrzejak, A., Reinefeld, A., Schintke, F., Schütt, T., Mastroianni, C., Fragopoulou, P., Kondo, D., Malecot, P., Cosmin Silaghi, G., Moura Silva, L., Trunfio, P., Zeinalipour-Yazti, D., Zimeo, E.: Grid architectural issues: State-of-the-art and future trends. CoreGRID White Paper (2008)
3. Arora, M., Das, S.K., Biswas, R.: 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, pp. 499–505. IEEE Computer Society, Washington, DC, USA (2002)
4. Babaoglu O.; Marzolla, M.T.M.: Design and implementation of a p2p cloud system. Tech. rep., Department of Computer Science, University of Bologna, Italy (2011)
5. Baikerikar, J.A., Surve, S.K., Prabhu, S.U.: Comparison of load balancing algorithms in a grid. Data Storage and Data Engineering, International Conference on pp. 20–23 (2010)
6. Baumgart, I., Heep, B., Krause, S.: OverSim: A flexible overlay network simulation framework. In: Proceedings of 10th IEEE GI/INFOCOM 2007, Anchorage, AK, USA, pp. 79–84 (2007)
7. Bing-jue, S., Kai-jun, W.: Research on cloud computing application in the peer-to-peer based video-on-demand systems. In: Intelligent Systems and Applications (ISA), 2011 3rd International Workshop on, pp. 1 –4 (2011)
8. Bonabeau, E., Dorigo, M., Theraulaz, G.: Swarm intelligence: from natural to artificial systems. Oxford University Press, Inc., New York, NY, USA (1999)
9. Brocco, A.: Overswarm: a simulation tool for biologically inspired peer-to-peer networks. Tech. Rep. TM-2011-4, Institute of Telematics, Karlsruhe Institute of Technology (2011)
10. Brocco, A.: ozmos: bio-inspired load balancing in a chord-based p2p grid. In: Proceedings of the 3rd workshop on Biologically inspired algorithms for distributed systems, BADS '11, pp. 9–16. ACM, New York, NY, USA (2011)
11. Brocco, A., Malatras, A., Huang, Y., Hirsbrunner, B.: Aria: A protocol for dynamic fully distributed grid meta-scheduling. ICDCS 2010 pp. 86–95 (2010)
12. Cao, J.: Self-organizing agents for grid load balancing. In: Proceedings of the 5th IEEE/ACM International Workshop on Grid Computing, GRID '04, pp. 388–395. IEEE, Washington, DC, USA (2004)
13. Chakravarti, A.J., Baumgartner, G.: The organic grid: Self-organizing computation on a peer-to-peer network. In: ICAC '04: Proceedings of the First International Conference on Autonomic Computing, pp. 96 – 103. IEEE Computer Society, Washington, DC, USA (2004)
14. Christodoulopoulos, K., Sourlas, V., Mpakolas, I., Varvarigos, E.: A comparison of centralized and distributed meta-scheduling architectures for computation and communication tasks in grid networks. Comput. Commun. **32**, 1172–1184 (2009)
15. Cunsolo, V.D., Distefano, S., Puliafito, A., Scarpa, M.: From volunteer to cloud computing: cloud@home. In: Conf. Computing Frontiers, pp. 103–104 (2010)
16. Dean, J., Ghemawat, S.: Mapreduce: simplified data processing on large clusters. Commun. ACM **51**, 107–113 (2008)
17. Dorigo, M., Stützle, T.: Ant Colony Optimization. Bradford Company, Scituate, MA, USA (2004)
18. Dörnemann, K., Prenzer, J., Freisleben, B.: 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, pp. 7:1–7:8. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), ICST, Brussels, Belgium, Belgium (2007)
19. Ferguson, D.F., Nikolaou, C., Sairamesh, J., Yemini, Y.: Economic models for allocating resources in computer systems pp. 156–183 (1996)
20. Folding@home: Folding@home distributed computing (accessed (April 5. 2012)). Http://folding.stanford.edu/
21. Fölling, A., Grimme, C., Lepping, J., Papaspyrou, A.: Decentralized grid scheduling with evolutionary fuzzy systems. In: Job Scheduling Strategies for Parallel Processing. Springer Verlag (2009). Lect. Notes Comput. Sci. vol. 5798

22. Forestiero, A., Leonardi, E., Mastroianni, C., Meo, M.: Self-chord: A bio-inspired p2p framework for self-organizing distributed systems. IEEE/ACM Trans. Netw. **18**(5), 1651–1664 (2010)
23. Foster, I., Kesselman, C.: Globus: A metacomputing infrastructure toolkit. The International Journal of Supercomputer Applications and High Performance Computing **11**(2), 115–128 (1997)
24. Foster, I.T., Iamnitchi, A.: On death, taxes, and the convergence of peer-to-peer and grid computing. In: M.F. Kaashoek, I. Stoica (eds.) Peer-to-Peer Systems II, Second International Workshop, IPTPS 2003, Berkeley, CA, USA, February 21-22,2003, Revised Papers, *Lecture Notes in Computer Science*, vol. 2735, pp. 118–128. Springer (2003)
25. Francesco Palmieri, D.C.: Swarm-based Distributed Job Scheduling in Next-Generation Grids. Springer (2007)
26. Gupta, R., Sekhri, V., Somani, A.K.: Compup2p: An architecture for internet computing using peer-to-peer networks. IEEE Trans. Parallel Distrib. Syst. **17**, 1306 –1320 (2006)
27. Haynie, D.: Biological thermodynamics. No. v. 424 in Biological Thermodynamics. Cambridge University Press (2001)
28. Huang, P.J., Yu, Y.F., Lai, K.C., Yang, C.T.: Distributed adaptive load balancing for p2p grid systems. In: Pervasive Systems, Algorithms, and Networks (ISPAN 2009), pp. 696–700 (2009)
29. Lu, K., Subrata, R., Zomaya, A.: An efficient load balancing algorithm for heterogeneous grid systems considering desirability of grid sites. In: Performance, Computing, and Communications Conference, IPCCC 2006, pp. 320–329 (2006)
30. Mell, P., Grance, T.: The nist definition of cloud computing. National Institute of Standards and Technology **53**(6), 50 (2009)
31. Milojicic, D.S., Kalogeraki, V., Lukose, R., Nagaraja, K., Pruyne, J., Richard, B., Rollins, S., Xu, Z.: Peer-to-peer computing. Tech. rep., HP Labs (2003)
32. Moallem, A., Ludwig, S.A.: Using artificial life techniques for distributed grid job scheduling. In: Proceedings of the 2009 ACM symposium on Applied Computing, SAC '09, pp. 1091–1097. ACM, New York, NY, USA (2009)
33. Montresor, A.: Anthill: a framework for the design and analysis of Peer-to-Peer systems. In: Proceedings of the 4th European Research Seminar on Advances in Distributed Systems. Bertinoro, Italy (2001)
34. Montresor, A., Meling, H., Babaoğlu, O.: Messor: Load-Balancing through a swarm of autonomous agents. In: In Proceedings of 1st Workshop on Agent and Peer-to-Peer Systems, pp. 125–137 (2002)
35. Murata, Y., Inaba, T., Takizawa, H., Kobayashi, H.: Implementation and evaluation of a distributed and cooperative load-balancing mechanism for dependable volunteer computing. In: DSN, pp. 316–325 (2008)
36. Nudd, G.R., Kerbyson, D.J., Papaefstathiou, E., Perry, S.C., Harper, J.S., Wilcox, D.V.: Pace–a toolset for the performance prediction of parallel and distributed systems. Int. J. High Perform. Comput. Appl. **14**(3), 228–251 (2000). DOI 10.1177/109434200001400306. URL http://dx.doi.org/10.1177/109434200001400306
37. Poli, R., Kennedy, J., Blackwell, T.: Particle swarm optimization. Swarm Intelligence **1**(1), 33–57 (2007)
38. Romberg, M.: The unicore architecture: Seamless access to distributed resources. In: Proceedings of the 8th IEEE International Symposium on High Performance Distributed Computing, HPDC '99, pp. 44–. IEEE Computer Society, Washington, DC, USA (1999). URL http://dl.acm.org/citation.cfm?id=822084.823237
39. Salehi, M.A., Deldari, H.: Grid load balancing using an echo system of intelligent ants. In: Proceedings of PDCN'06, pp. 47–52. Anaheim, CA, USA (2006)
40. Schopf, J.M.: Ten actions when Grid scheduling: the user as a Grid scheduler. Kluwer Academic Publishers, Norwell, MA, USA (2004)
41. Shah, R., Veeravalli, B., Misra, M.: 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), 1675 –1686 (2007)
42. Stoica, I., Morris, R., Karger, D., Kaashoek, F.M., Balakrishnan, H.: Chord: A scalable peer-to-peer lookup service for internet applications. In: SIGCOMM '01, vol. 31, pp. 149–160. ACM Press, New York, USA (2001)
43. Varga, A.: The omnet++ discrete event simulation system. Proceedings of the European Simulation Multiconference (ESM'2001) (2001)

44. Wu, W., Chen, Y., Zhang, X., Shi, X., Cong, L., Deng, B., Li, X.: Ldht: Locality-aware distributed hash tables. In: Information Networking, 2008. ICOIN 2008. International Conference on, pp. 1–5 (2008)