WE-AMBLE: A WORKFLOW ENGINE TO MANAGE AWARENESS IN COLLABORATIVE GRID ENVIRONMENTS

Pilar Herrero[1]
José Luis Bosque[2]
Manuel Salvadores[1]
María S. Pérez[1]

Abstract

Grid computing shares heterogeneous resources in dynamic service-based environments. This kind of system has the major advantage of enabling rapid composition of distributed applications. However, equilibrating the amount of work assigned to each of the nodes in a grid environment is a complex problem, even more so than for other kinds of parallel systems. In this paper, we present a new extension and reinterpretation of one of the most successful models of awareness in Computer Supported Cooperative Work (CSCW), called the Spatial Model of Interaction (SMI), which manages awareness of interaction through a set of key concepts, to manage task delivery in collaborative distributed systems. This model, called AMBLE (Awareness Model for Balancing the Load in Collaborative Grid Environments), also applies some theoretical principles and theories of multi-agent systems to create a collaborative and cooperative environment that provides autonomous, efficient and independent management of resources available in a grid environment. WE-AMBLE, a Workflow Engine to manage awareness in collaborative grid environments through the AMBLE concepts, provides a workflow engine to manage different levels of awareness, allowing different virtual organizations to share computational resources based on open protocols and interfaces.

Key words: workflow, collaborative environments, awareness management, delivery task, grid environments

1 Introduction

Grid computing shares heterogeneous resources in dynamic environments. The complexity of achieving this goal is caused by several factors: the existence of different virtual organizations, the dynamism of the underlying architecture, and the heterogeneity of the involved resources are some of the most challenging aspects.

In order to provide better capabilities on a grid, it is essential to manage the resources with a workflow engine which will take the appropriate, and complex, decisions about the allocation of processes to the resources of the system. Resource management includes other tasks, such as resources discovery, resources registration, and monitoring. The resource manager is expected to achieve load balancing within the grid.

Equilibrating the amount of work assigned to each node in a grid is a complex problem, even more so than for other kinds of parallel systems. Even though load balancing has received a considerable amount of interest, it is still not completely solved (Harchol-Balter and Downey 1997; Das, Harvey, and Biswas 2001; Zomaya and Teh 2001; Xiao, Chen, and Zhang 2002). Nevertheless, this problem is central for minimizing the application's response time, optimizing the exploitation of resources and avoiding overloading some processors while others are idle. Grids present additional challenges, since they can easily become heterogeneous, requiring load distributions that take into consideration each node's computational features as well as the services that each node offers to the grid – each node could offer different services. In order to provide flexible and efficient load-balancing mechanisms, new technologies could be applied.

This is the case for multi-agent systems, an already mature technology, which offers promising features to resource managers. The reactivity, proactivity and autonomy, as essential properties of agents, can help in the complex task of managing resources in dynamic and changing environments. Additionally, the cooperation among agents, which interchange information and resources status, allows load balancing mechanisms to be performed and efficiently deployed on a grid. In this sense, these mechanisms have common goals with current collaborative systems, and several synergies between both disciplines can arise.

In this paper, we present a new extension and reinterpretation of the Spatial Model of Interaction (SMI), an

[1]FACULTAD DE INFORMÁTICA, UNIVERSIDAD POLITÉCNICA DE MADRID, CAMPUS DE MONTEGANCEDO S/N, 28.660 BOADILLA DEL MONTE, MADRID, SPAIN

[2]DEPARTAMENTO DE ELECTRÓNICA Y COMPUTADORES, FACULTAD DE CIENCIAS, UNIVERSIDAD DE CANTABRIA, AV DE LOS CASTROS S/N, 39.005 SANTANDER, SPAIN, (JOSELUIS.BOSQUE@UNICAN.ES)

The International Journal of High Performance Computing Applications, Volume 22, No. 3, Fall 2008, pp. 250–267
DOI: 10.1177/1094342007086225
© 2008 SAGE Publications Los Angeles, London, New Delhi and Singapore
Figures 1, 3, 6–9 appear in color online: http://hpc.sagepub.com

Downloaded from hpc.sagepub.com by guest on February 25, 2013

abstract awareness model designed to manage awareness of interactions in cooperative applications through a set of key concepts. The SMI is based on a set of key concepts which are abstract and open enough as to be reinterpreted in many other contexts with very different meanings (Greenhalgh 1999). Thus, this paper presents a new reinterpretation of this model, and its key concepts, called AMBLE (Awareness Model for Balancing the Load in Collaborative Grid Environments), in the context of an asynchronous collaboration in a grid environment. This reinterpretation merges research carried out in CSCW, agents, and grid communities, to create a collaborative and cooperative agents-based grid environment.

WE-AMBLE, a Workflow Engine to manage awareness in collaborative grid environments through the AMBLE concepts, has been designed from the beginning to be a parametrical, generic, open, model that could be extended and adapted easily to new ideas and purposes. This model allows management not just of resources and information but also of interaction and awareness. WE-AMBLE allows: i) control of the user interaction (through the aura concept); ii) guiding the awareness toward specific users and resources (through both focus and nimbus concepts); iii) scaling interaction through the awareness concept. This model has also been designed to apply successful agent-based theories, techniques and principles to deal with resources sharing as well as resources assignment inside the grid.

Following some of the main OGSA (Foster et al. 2002) characteristics our proposal provides an open interface. In this way, WE-AMBLE has the ability of being: i) extended and adapted to new modifications in the model; ii) scalable to different configurations; iii) re-used to manage different levels of awareness in different infrastructures and virtual organizations; iv) free of bottlenecks because of its distributed nature.

2 Related Work

Two topics are closely related to this work. Firstly, the main achievements of the collaborative systems, and more specifically the negotiation, have been investigated and applied as load-balancing strategies. Most of these approaches use multi-agent systems as essential components for achieving load balancing in distributed systems. Secondly, although grids could be considered as specific distributed systems, there are important characteristics in grids that influence in the relationship between agents and grids.

Different load-balancing implementations are being used in grid environments. For instance, the Satin system, which is intended for divide-and-conquer grid applications, uses a load-balancing algorithm named Cluster-aware Work Stealing (CRS; van Nieuwpoort et al. 2006). CRS runs applications in grid environments almost as efficiently as on local clusters. The AppLeS (Application-level Scheduling) project (Berman et al. 1996) provides a scheduling framework for grid applications. This project selects the best set of resources for a specific application from a pool of grid resources. Unlike these implementations, our system applies the awareness concept with the aim of improving the load balance of a grid environment.

However, we would like to highlight the relevant position that grid workflows have in the grid community. A huge number of works have appeared in the literature about grid workflows, for example: Tannenbaum et al. (2002), Deelman et al. (2004), Oinn et al. (2004), von Laszewski et al. (2004), and Chen and Yang (2006a, 2006b). Condor DAGMan (Directed Acyclic Graph Manager; Tannenbaum et al. 2002) is a meta-scheduler for jobs running in Condor, a resource management system intended for compute-intensive jobs, which has been developed at the University of Wisconsin-Madison. DAGMan uses directed acyclic graphs in order to represent dependencies among jobs. Pegasus (Deelman et al. 2004) is a workflow management system used within the GriPhyN project, whose aim is to give support to large volumes of data in physics experiments. Pegasus maps abstract workflows to a subset of available grid resources, producing an executable workflow. Taverna (Oinn et al. 2004) is the workflow manager of the myGrid project. This manager provides a workflow language SCUFL (Simple Conceptual Unified Flow Language) and a graphical interface to make easier the development and running of bioinformatics workflows over distributed computer resources. GridAnt (von Laszewski et al. 2004) is a user-controllable workflow manager developed at the Argonne National Laboratory. It uses the workflow engine of Apache Ant, providing grid tasks to be used in the Ant framework.

A complete taxonomy and the classification of different workflow systems in this taxonomy are described by Yu and Buyya (2005). This taxonomy is constructed according to the main functions and architectures of the grid workflow systems.

Moreover, some interesting results in this area have already appeared, for example Chen and Yang (2006a, 2006b).

Another important related aspect is the negotiation. The negotiation takes an important role in agent systems. Four different negotiation models were studied by Shen et al. (2002) for agent-based load balancing and grid computing: contract net protocol, auction model, game theory based model and discrete optimal control model.

Both grid and agent solutions have been developed for distributed systems, although every discipline has its own concerns and goals. Challenges associated with the inter-

Downloaded from hpc.sagepub.com by guest on February 25, 2013

action between grids and agents have been clearly defined by Foster et al. (2004). Among others, the most important lines of research identified for the overlapping of both fields are: autonomous services, dynamic and stateful services (Greenhalgh and Benford 1995); negotiation and Service Level Agreements (SLA); virtual organization management (Cheung et al. 2004); and security.[1]

As an example of the successful combination of grid and agents, a real grid system has been built by means of mobile agent technology. SMAGrid (Strong-Mobile Agent-Based Grid; Zhang and Luo 2003) is composed of multiple agents used for assigning resources to tasks. SMAGrid is based on Tagent (Letsch 2000), a mobile agent system which follows the MASIF standard and provides strong mobility. Other works that use agents in grid environments have been described previously in this section.

As was remarked in the introduction, what really makes the difference with the above-mentioned systems is the ability to manage not just resources and information but also interaction and awareness. WE-AMBLE applies successful agent-based theories, techniques and principles to deal with resource sharing as well as resource assignment inside the grid. Additionally, the WE-AMBLE implementation is scalable, allows management of different levels of awareness and is free of bottlenecks because of its distributed nature.

3 The Spatial Model of Interaction (SMI)

The Spatial Model of Interaction (SMI), defined for application to any Computer Supported Cooperative Work (CSCW) system where a spatial metric can be identified, has been driven by a number of objectives (Benford and Fahlén 1993):

- Scalability: It is based on the concept of *aura*. Each object has an aura for each medium in which it can interact, because the aura defines the volume of space within which this interaction is possible (Greenhalgh 1999).
- Interactions: The SMI assumes a space populated by potentially communicating objects. The SMI provides a framework for these objects to manage their interaction, and communication between every pair of objects (Greenhalgh and Benford 1995).

The model itself defines five linked concepts: medium, focus, nimbus, aura and awareness. These are extended by the additional concepts of adapters and boundaries.

Medium: A prerequisite for useful communication is that two objects have a compatible medium in which both objects can communicate.

Aura: Defined as the sub-space which effectively bounds the presence of an object within a given medium and which acts as an enabler of potential interaction (Fahlén and Brown 1992). Once aura has been used to determine the potential for object interactions the objects themselves are subsequently responsible for controlling these interactions.

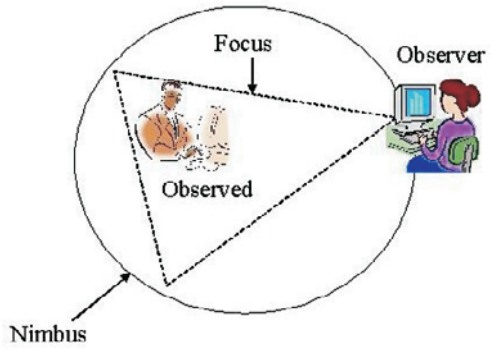

Fig. 1 Key concepts in the spatial model of interaction.

In each particular medium, it is possible to delimit the observing object's interest. This idea was introduced by S. Benford in 1993 as and was called *focus*. In the same way, it is possible to represent the observed object's projection in a particular medium. This area is called *nimbus* (see Figure 1).

Awareness: Quantifies the degree, nature or quality of interaction between two objects. One object's awareness of another object quantifies the subjective importance or relevance of that object. Awareness between objects in a given medium is manipulated via *focus* and *nimbus*, requiring a negotiation process. Considering, for example, A's awareness of B, the negotiation process combines the observer's (A's) focus and the observed's (B's) nimbus. For a simple discrete model of focus and nimbus, there are three possible classifications of awareness values when two objects are negotiating unidirectional awareness (Greenhalgh 1999):

- *Full awareness:* The awareness that object A has of object B in a medium M is "full" when object B is inside A's focus and object A is inside B's nimbus.
- *Peripheral awareness:* The awareness that object A has of object B in a medium M is "peripheral" when object B is outside A's focus but object A is inside B's nimbus, or object B is inside A's focus but object A is outside B's nimbus.
- *No awareness:* An object A has no awareness of object B in a medium M when object B is outside A's focus and object A is outside B's nimbus.

Downloaded from hpc.sagepub.com by guest on February 25, 2013

4 AMBLE: Reinterpreting the Key Awareness Concepts

Let us consider a system containing a set of nodes $\{n_i\}$ and a task T that requires a set of processes to be solved in the system. Each of these processes necessitates some specific requirements (such as disk space, speed of CPU, programs, database connectivity and so on), being r_i the set of requirements associated to the process p_i, and therefore each of the processes will be identified by the tuple (p_i, r_i) and T could be described as $T = \sum_i \{(p_i, r_i)\}$.

The AMBLE model takes into account that one of the major goals of grid computing is increasing the collaboration capabilities of the system, starting with a simple, abstract and preliminary interpretation of the SMI key concepts in the context of an asynchronous collaboration. Thus the AMBLE model proposes an awareness infrastructure, based on these concepts, that is capable of managing the load management of grid environments. This model reinterprets the SMI key concepts as follows:

Focus: Is interpreted as the subset of the space on which the user has focused his attention with the aim of interacting with it. Regarding tasks, and given a node n_i in the system requiring the execution of a given task (T), the focus of this node would contain at least the subset of nodes that are composing the virtual organization (VO) in which this node is involved.

$$\begin{aligned} \text{Focus: Node} &\rightarrow \text{System} \\ n_i &\rightarrow \{n_j\} \end{aligned}$$

The focus will be delimited by the *aura* of the node in the system. The focus of a given node could be modified and oriented towards other VOs if needed.

Nimbus: Is defined as a tuple (Nimbus = (*NimbusState, NimbusSpace*))containing information about: (a) the load of the system at a given time (*NimbusState*); (b) the subset of the space in which a given node projects its presence (*NimbusSpace*).

The *NimbusState* concept will depend on the processor characteristics as well as on the load of the system at a given time. In this way, the *NimbusState* could have three possible values: *Null*, *Medium* or *Maximum*. If the load of a given node is not high, and this node could receive some processes (p_i) or even the whole task T, its *NimbusState* will get the *Maximum* value. If there are no receptors nodes in the system and a given node n_i can accept, at least, a process, then the *NimbusState* of this node n_i would be *Medium*. Finally, if the node n_i is overloaded, its nimbus would be *Null*. The *NimbusSpace* will determine those machines that could be included in the task assignment process. The *NimbusSpace* will be delimited by the *aura* (bounding the effective and potential area of interaction) of the node in the system.

Awareness of Interaction (AwareInt): This concept will quantify the degree, nature or quality of asynchronous interaction between distributed resources. Following the awareness classification introduced by Greenhalgh (1999), this awareness could be *Full*, *Peripheral* or *Null*.

$$AwareInt(n_i, n_j) = Full$$

$$n_j \in Focus(\{n_i\}) \wedge n_i \in Nimbus(n_j)$$

The awareness of interaction will be peripheral if

$$AwareInt(n_i, n_j) = Peripheral$$

$$n_j \in Focus(\{n_i\}) \wedge n_i \notin Nimbus(n_j)$$

or

$$n_j \notin Focus(\{n_i\}) \wedge n_i \in Nimbus(n_j)$$

The AMBLE model is more than a reinterpretation of the SMI, it extends the SMI to introduce some new concepts such us:

Interactive Pool: This function returns the set of nodes $\{n_j\}$ interacting with the n_i node at a given moment. Given a system and a task T to be executed in the node n_i

$$\begin{aligned} InteractivePool: System &\rightarrow System \\ \{n_i\} &\rightarrow \{n_j\} \end{aligned}$$

$$AwareInt(n_i, n_j) = Full \text{ then } n_j \in InteractivePool(n_i)$$

Task Resolution: This function determines if there is a service in the node n_i, being NimbusState(n_i)/=Null, that could be useful for executing the task T (or at least one of its processes).

$$\begin{aligned} n_i = \sum_i \{s_i\} \; TaskResolution: Node \times Task &\rightarrow Task \\ \{n_i\} \times T &\rightarrow \{p_i, s\} \end{aligned}$$

where s is the "score" to execute p_i in n_i node, being its value within the range $[0, \infty)$: 0 if the node n_i fulfills all the minimum requirements to execute the process p_i; once the node n_i fulfills the all the minimum requirements to execute the process p_i, the higher the surplus over these requirements, the higher will be the value of this score.

This concept would also complement the nimbus concept, because the *NimbusSpace* will determine those

Downloaded from hpc.sagepub.com by guest on February 25, 2013

machines that could be included in the task assignment process because they are not overloaded. This only means that they could receive more workload, but the task T or at least one of its processes p_i will be executed in n_i if, an only if, there is a service s_i in the node n_i that could be useful to execute any of these p_i processes.

Collaborative Organization: This function will take into account the set of nodes determined by the *Interactive Pool* function and will return those nodes of the system which are most suitable for executing the task T (or at least one of its processes p_i). This selection will be made by means of the *TaskResolution* function.

$$CollaborativeOrganization: System \times Task \longrightarrow System$$

$$\{n_i\} \times T \longrightarrow \{n_j\}$$

5 Load Balancing Algorithm in AMBLE

In this section we will describe the load balancing algorithm introduced in the AMBLE awareness model, and how it will be applied to our distributed and collaborative multiagent architecture in grid environments. The main characteristics of this algorithm are that it is dynamic, distributed, global, and takes into account the system heterogeneity.

Even though currently there is a strong polemic about user and non-user interruption algorithms (Russ et al. 1998; Harchol-Balter 2001), in this paper we deal with a non-interruption algorithm and therefore when the task's execution is started in a node, this execution has to finish in the same node.

Generally a dynamic load balancing algorithm consists of four policies: a state/load measurement rule, an information exchange rule, an initiation rule and a load balancing operation (Xu and Lau 1997).

5.1 State Measurement Rule

This rule will obtain information about the computational capabilities of the node in the system. This information, quantified by a load index, provides awareness of the NimbusState of the node. This dynamic index should be frequently measured, and therefore it should provide a good estimation of a node computing capabilities. The choice of a load index has a huge impact on load balancing efficiency (Kunz 1991).

5.1.1 Load index The load index calculation is performed by the benchmark agent. In this paper the load index is evaluated based on two parameters (one static and one dynamic):

- Node computational power (P), which depends on the node computational architecture and takes into account CPU, memory and I/O features. It will function as a static parameter.
- The CPU assignment which is defined as the percentage of CPU time that would be available to a newly created task, on a specific node. It will function as a dynamic parameter.

The benchmark agent will implement the load measurement rule, measuring periodically the required parameters and evaluating the load-index of every node "n_i", belonging to the grid, based on the following formula:

$$\text{load-index}_i = p_i / (p_{max} * np_i)$$

where

- p_i: represents n_i's Linpack benchmark (University of Tennessee 2006; see Section 10)
- p_{max} : represents the Linpack of the best node
- np_i: represents the number of processes that are executing in the node "n_i" at a given moment (this model only considers CPU-intensive tasks).

In order to calculate the NimbusState of a given node, the benchmark agent monitors the state of the system by applying the previous formula. The value of the node's NimbusState is calculated by the function *NimbusCal* as follows:

$$NimbusCal = \begin{cases} Null \Rightarrow load\text{–}index_i < 0.33 \\ Medium \Rightarrow 0.33 \leq load\text{–}index_i \leq 0.66 \\ Maximum \Rightarrow load\text{–}index_i > 0.66 \end{cases}$$

The *NimbusCal* function returns a value close to zero when the load of the node is very high. The closer to 1 this value is, the lower is the load of the node.

5.1.2 NimbusState The NimbusState of the node will be determined by the load index and it will depend on the node capacity at a given time. This state determines if the node could execute more (local or remote) tasks. Its possible values would be:

- *Maximum*: The load index is low and therefore this infra-utilized node will execute all the local tasks, accepting all new remote execution requests coming from other nodes. The NimbusState will decrease while load index increases.
- *Medium*: The load index has an intermediate value and therefore the node will execute all the local tasks, interfering in load balancing operations only if there are no other nodes in the system whose NimbusState would be maximum.

Downloaded from hpc.sagepub.com by guest on February 25, 2013

- *Null*: The load index has a high value and therefore the node is overloaded. In this situation, the node will not execute new tasks, the computational capabilities available are very low and for that reason it will reject any request for new remote execution. If the node receives any request for local execution, it will start a new load-balancing operation that could be successful, reallocating the new task execution in the appropriate node, or unsuccessful, keeping the task in the queue until any of the nodes changes its state.

The NimbusState of the node depends on the load index value and an increase or decrease of this index over a specific threshold will imply the corresponding modification in the NimbusState.

5.2 Information Exchange Rule

The knowledge of the global state of the system will be determined by a policy on the information exchange and therefore it is important that this policy is suitable for the problem which is being solved. This policy should keep the information coherence without overloading the network with an excessive number of unnecessary messages. Traditionally there are three well known and widely used rules (Xu and Lau 1997).

An optimum information exchange rule for the AMBLE model should be based on events (Beltrán, Bosque, and Guzmán 2004). This rule only collects information when a change in the Nimbus (in the *NimbusState* or in the *NimbusSpace* or in both) of the nodes is made. If the latter, the node that has modified its nimbus will be in charge of notifying this modification to the rest of the nodes in the system, thus avoiding synchronization points.

5.3 Initiation Rule

The initiation rule determines when to begin a new load balancing operation. The execution of a load balancing operation incurs non-negligible overhead, and therefore the expected performance benefit must outperform this overhead. An initiation policy is thus needed to determine whether a balancing operation will be profitable. Therefore the initialization rule has a strong impact in the algorithm performance.

As the model implements a non-user interruption algorithm, the selection of the node must be made just before sending the task execution. Once the execution of the process starts in a specific node it has to finish in the same node.

The decision of starting a new load balancing operation is completely local, depending on the local information storage. If an overload node receives a new task $T = \sum_i \{(p_i, r_i)\}$ to be executed, and it cannot execute it (*NimbusState*(n_i) = Null or *NimbusState*(n_i) = Medium), the process load balancing operation will be automatically launched to allocate the task (or at least some of its processes) in those nodes that are inside its focus if $Focus(n_i) \neq \{\varnothing\}$. Otherwise, the task will pass to the node's queue waiting for a change on *NimbusState* or on the node's Focus.

5.4 Load Balancing Operation

Once the node has made the decision to start a new load balancing operation, this operation will be divided into another three different rules: localization, distribution and selection. Each of them will be described in the following subsections.

5.4.1 Localization rule The *localization rule* determines the partners of the balancing operation, i.e. the processors to be involved in the operation. Given a task $T = \sum_i \{(p_i, r_i)\}$ to be executed in the node n_i, and taking into account that *NimbusState*(n_i) = Null or *NimbusState*(n_i) = Medium, the localization rule has to determine which nodes are involved in the *CollaborativeOrganization* of the node n_i.

In order to make it possible, firstly, the AMBLE model will need to determine the awareness of interaction of this node with those nodes inside its focus. To optimize the implementation, the previous awareness values are dynamically updated based on the information exchange rule. Those nodes whose awareness of interaction with n_i was Full will be part of the Interactive Pool of n_i to solve the task T, and from that pre-selection the TaskResolution method will determine those nodes that are suitable for efficiently solving the task in the environment.

5.4.2 Selection and distribution rule This algorithm joins the selection and distribution rules because it determines which nodes (among all the nodes constituting the *CollaborativeOrganization*) will be in charge of executing each of the processes making up the task T. The proposed algorithm takes into account the *NimbusState* of each of the nodes – only those whose *NimbusState* would be maximum or medium could receive new processes – as well as the *TaskResolution* to solve any of T's processes.

The goal of this algorithm is to find the most equilibrated assignment of processes to computational nodes. The optimal assignment of these processes, analyzing all the possible combinations, is an NP-complete problem which cannot be solved in a reasonable computational time. Hence this algorithm is based on a set of heuristics which result in an equilibrated spread even though it may not be the optimum. This spread is made in an iterative way. Firstly, a complete distribution is made taking into account all the processes making up the task T as well as all

Downloaded from hpc.sagepub.com by guest on February 25, 2013

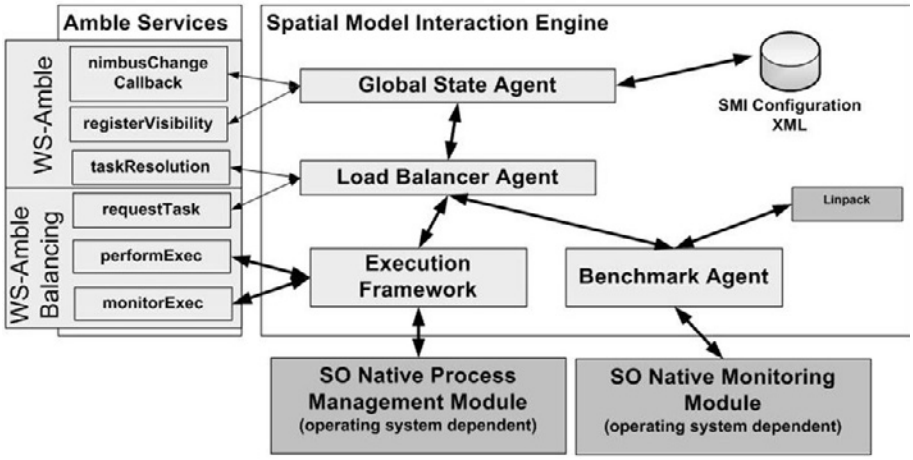

Fig. 2 The load balancing model: middleware architecture.

the computational nodes implicated in the *CollaborativeOrganization.* If, in this first iteration, it is be possible to assign all the processes to be distributed to one of the nodes involved in the *CollaborativeOrganization*, the algorithm will have finished. Otherwise, it will again calculate the *NimbusState* of the nodes belonging to the *CollaborativeOrganization*, repeating the complete process. The sequence of steps that implements the assignment heuristic is:

1. From all the nodes belonging to the *CollaborativeOrganization,* only those whose *NimbusState* Null will be selected in the first instance.
2. If any of the processes $p_i \in T$ could be executed in just one of the nj nodes selected in the previous step, this process would be automatically assigned to the node that can execute it.
3. Among all the remaining nodes, and for each and every process, the node whose score was highest will be selected to execute the corresponding process.
4. Once a T process had been assigned to every node inside the pool, a message will be sent to each of these nodes requiring the execution of the designated processes. The remote nodes can accept the execution; in this case, the process will be definitely assigned, and, once the process execution has started, they should again evaluate its *NimbusState* returning its value to the origin node. However, if the candidate node rejects the process execution, because of latency problems (the candidate node has changed its *NimbusState* to Null but the origin node doesn't know it yet), the origin node should look for a new candidate to assign the corresponding process.
5. Repeat the previous steps (1–4) until all the processes have been assigned or until all the nodes in *CollaborativeOrganization* have *NimbusState* = *Null*.
6. Once all the nodes in *CollaborativeOrganization* have achieved a NimbusState = Null or once it is impossible to find a candidate in the *CollaborativeOrganization* to execute any of T's processes, the task will be waiting in n_i for a change in the state of any of the nodes that comprise the *CollaborativeOrganization* (n_i) (or even a change in its own state) or an aura extension could be made (thereby extending its focus or its nimbus) with the aim of repeating the complete process from the beginning

6 WE-AMBLE Architecture

From the beginning WE-AMBLE has been designed to avoid any possible bottlenecks. In fact, the model architecture has been designed in a distributed way. In the following figure (Figure 2), it is possible to appreciate that the middleware architecture of the load-balancing model has been associated to a single node. This architecture is replicated (according to Salvadores et al. 2005) in all the nodes of the grid to avoid any possible bottlenecks.

The above architecture has been separated into three different parts:

- **SMI-Engine (Spatial Model of Interaction Engine)**: This is the main core of the architecture and contains

Downloaded from hpc.sagepub.com by guest on February 25, 2013

those components that implement all the logic of the SMI and the load balancing algorithms explained in Section 6. This engine is made up of the following modules:

- Benchmark agent: This agent, based on the Linpack benchmark (University of Tennessee 2007), maintains a performance measure of the node which can be evaluated from the load balancer. Moreover, this agent is connected to the monitor module of the system to evaluate its availability.
- Global state agent: This agent compiles all the information related to the two main concepts of the SMI (*Focus* and *Nimbus*), providing information about those nodes that are available in the system to carry out a load-balancing operation at a given moment.
- Load balancer agent: This module implements the logic for the load-balancing operation, according to the scores that each of the nodes has achieved executing a specific process, and makes the final decision of which node will execute each process.
- Execution framework: This interface contains the modules dependent on the operating system (OS) to access the process management APIs.
- SO native process management: This depends on the OS and uses those functionalities that the APIs of this OS offer to supply the process execution.
- SO native monitoring module: This module also depends on the OS and uses those functionalities that the APIs of this OS offer to monitor the state of the computer.

- **AMBLE-Service**: A Web service[2] interface that provides those methods necessary to establish the communication between nodes through SOAP[3] messages. The operations deployed on this service are:
- registerVisibility: When a node detects a new resource inside its focus, it invokes this operation. Moreover, if the observer node is also inside the observed NimbusSpace, it would be included in the awareness of interaction record with a value equal to Full.
- nimbusChangeCallback: This operation receives the changes that a specific node has on its NimbusState.
- requestTask: This method is invoked by a client requiring the execution of the task T composed by n processes.
- taskResolution: This method is invoked by a node requiring "offers/scores" for the processes associated to a specific task.
- performTask: This method is invoked to order the process execution once the process has been assigned to a particular node.
- monitorExec: This operation is used to monitor the state of execution of a process in an identifiable node.

7 WE-AMBLE Workflow Step By Step

In Figure 3 it is possible to appreciate the sequence of operations unleashed inside a node after it has received the registerVisibility message from another one:

1. The first step is to corroborate if the node that invoked the operation registerVisibility is inside its NimbusSpace.
2. If the remote node is inside its NimbusSpace, (enableReg=true), the remote node will add it to its awareness of interaction record with a value equal to Full, and the origin node will return its NimbusState. This execution branch determines that both nodes could collaborate, if needed, to carry out a load-balancing operation.
3. However, if the remote node is not inside its NimbusSpace, it will return a rejected message. The remote node will include the origin node in its awareness of interaction record with a value equal to Peripheral. This awareness could be change to Full if the NimbusSpace is modified and the remote node is included inside.

Figure 4 shows the sequence of steps necessary to carry out a load-balancing operation in a node that requires delegation of the execution of several processes associated to the task T:

1. The node receives a message from the grid client containing the composition of processes to be executed. Before starting the load-balancing operation, it will calculate the *Interactive Pool.*
2. Once the origin node has this set of nodes, it will ask each of them (concurrently and through SOAP messages) for the score associated with each of the processes of the T task.
3. The origin node will set this information in a list, and this list will be the input for the resolveLoadBalance function (this function implements the final selection heuristic as has been defined in Section 6.4.2) and returns the assignment of the processes to the nodes.
4. For each of these assignments the origin node will send an execution message to the node. Although the remote node could reject the execution of the process (because of its NimbusState), it could happen that, because the system is asynchronous, its NimbusState changes to Null, thereby creating the possibility of accepting the process execution. If the latter, the executionId assigned is NOT_ASSIGNED and the process will be assigned in the next round (in which the aura would have been increased as well).

Downloaded from hpc.sagepub.com by guest on February 25, 2013

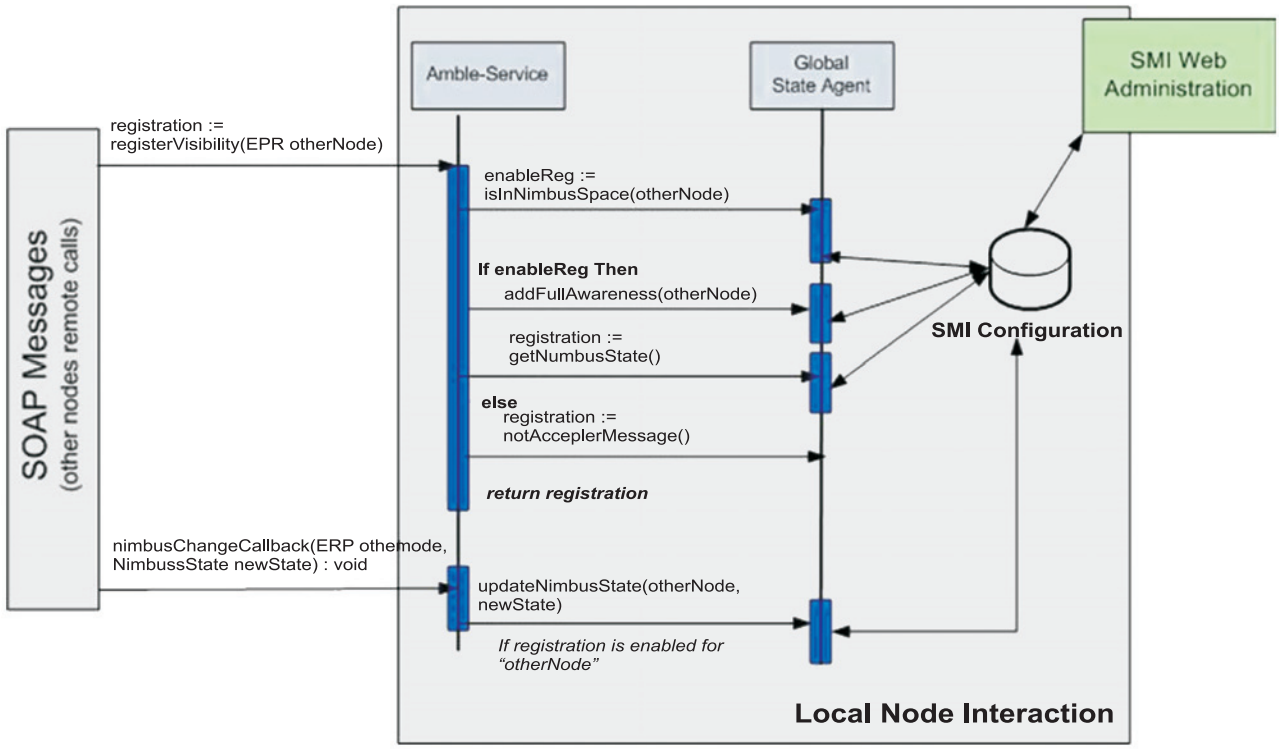

Fig. 3 General interaction diagram: AwareInt=Full and AwareInt=Peripheral.

The last situation will occur if any of the processes has not been assigned. In this case the SMI Engine will repeat the previous algorithm increasing the aura to catch those nodes that are located at a higher distance. This loop will be repeated until there are no more nodes in the system and, if so, those processes that have not been assigned will pass to the queue.

8 WE-AMBLE Architecture

WE-AMBLE provides an open interface to manage different levels of awareness, allowing different virtual organizations to share computational resources based on open protocols and interfaces.

As far as we know, none of the latest WS specifications offers functionalities useful for creating awareness models or offers specific functionalities to manage task-balancing delivery in collaborative grid environments, as WE-AMBLE does. Through WE-AMBLE, it will be possible to create new architectures oriented toward services. Figure 5 shows the different levels of functionality offered by each of WS_AMBLE components:

1. **WS-Addressing**: Allows a transport mechanism to address services and messages (W3C standard).
2. **WS-Resource Framework**: This specification, an evolution of the OGSI specification, has been recently standardized by the OASIS consortium. It promotes a standard way of deploying resources and how to consult about them.
3. **WS-Notification and WS-Topic**: Jointly, they provide the facility to establish mechanisms based on the model of interaction, publication and subscription.

WE-AMBLE uses standard mechanisms based on WS to interact with other resources. These mechanisms are supplied by the recently standardized WS-Resource Framework specification. In addition, the communication model, founded on publication/subscription, is based on the WS-N/WS-Topic specification. WS-N supplies the mechanism to subscribe two resource management nodes, in an AMBLE-grid environment, to the NimbusSpace as well as to the NimbusSpace, with emphasis on changes in state.

Downloaded from hpc.sagepub.com by guest on February 25, 2013

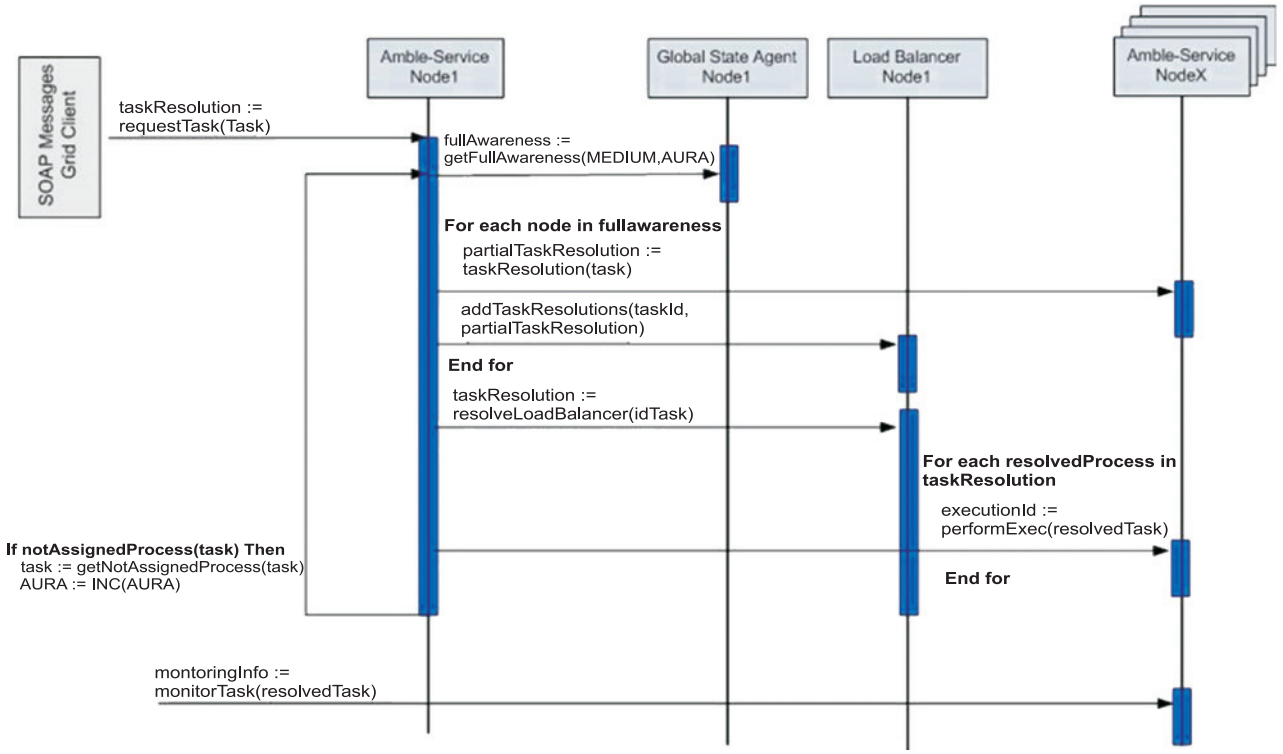

Fig. 4 General interaction diagram: load-balancing operation.

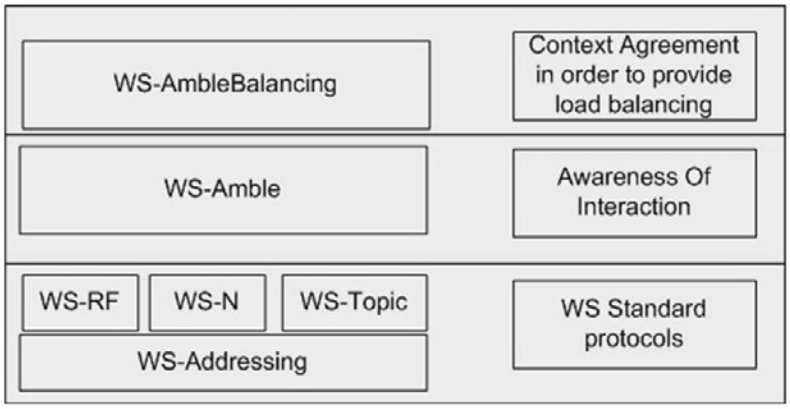

Fig. 5 WE-AMBLE architecture.

9 AMBLE as a Common Interface

The results presented in this paper come from the implementation of a first prototype to validate the model, measure its efficiency and calculate the overhead. But AMBLE goes beyond this because it allows the implementation of the interaction among different virtual organizations (VO); each VO could implement this inter-

Downloaded from hpc.sagepub.com by guest on February 25, 2013

action with the AMBLE model in a different way, using the management tools and the scheduling elements most suitable for each specific situation.[4]

A VO implements its own authorization mechanisms based on LDAP as well as the configuration elements for the SMI (NimbusState, NimbusSpace, aura, …) that are stored in a data-store. This VO could have some specific tools to manage these data structures. In the same way, another VO could have the SMI configuration based on agents in charge of modifying the AMBLE configuration according to the environment. AMBLE could establish non hierarchical interaction structures. The interaction structure associated to the organization will depend on the global configuration of the SMI. But WS-AMBLE also leaves the resources management open: one resource could be managed, through its configuration, by more than one VO.

10 Experimental Results

This section presents a set of experiments with the following objectives: to corroborate whether the AMBLE's tasks delivery works in a real and heterogeneous grid environment; to detect the overhead introduced by the AMBLE's model in a real environment; and to measure the AMBLE's speedup in different scenarios. These experiments, as well as the set of scenarios and tests presented, could be extended if the reviewers consider it appropriate.

10.1 The Grid Environment Infrastructure

The tests presented in this paper have been evaluated in a real and heterogeneous services-oriented grid environment. The system heterogeneity is reflected not just in the architecture of the computational nodes (and therefore in the computational power), but also in the OS utilized. The grid infrastructure was made up of 20 nodes with the following characteristics: 8 of them were Intel Centrino P4 1.5 GHz with 0.5 GB of memory (referred to here as "Type A" nodes), 11 of them were Intel P4 3.0 GHz (B) with 1 GB of memory (referred to as "Type B" nodes) and the last one was an Intel Xeon 3.6 GHz (C) with 4 GB of memory (referred to as "Type C" nodes).

In order to carry out the model evaluation we have selected a set of CPU-intensive tests based in iterations over the Linpack benchmark (University of Tennessee 2007). The High-Performance Linpack benchmark is a numerically intensive test which was developed in 1979 to measure the floating-point performance of computational systems and since then has been widely used in scientific environments for carrying out performance measuring experiments. The following subsection presents a comparison of the speedup achieved with the different types of resources (Types A, B and C) mentioned above, as well as the different scenarios raised to make this evaluation possible. In each of these scenarios, there is a task T, composed of 20 processes, to be executed and a node N that receives the T execution request. Each of these scenarios also presents four different tests; each of them differs from the others in the size of the processes to be executed.

Table 1
Response time (s).

	Type A	Type B	Type C
Test 1	4.95	3.86	3.48
Test 2	24.02	19.20	17.80
Test 3	47.00	38.72	34.97
Test 4	232.93	192.30	175.03

Table 1 presents the response times, in seconds, for each of the tests carried out in the different nodes of the grid.

The metric used to measure the AMBLE performance was the response time achieved for each of the tests in the proposed scenarios. These measures allow calculation of the speedup as well as the heterogeneous efficiency of the system (Pastor and Bosque 2001).

The heterogeneous efficiency depends on the total computational power of the grid, and this power represents the total amount obtained by adding the computational power of each of the nodes comprising the grid, which is the amount of work that the node can perform in a given interval (Pastor and Bosque 2001). The optimum time in the grid can be defined as the time that one computer – with a computational power equivalent to the total computational power of the grid – would invest to solve the problem. Additionally, the communication overload introduced by the AMBLE model is calculated.

10.2 Experimental Scenarios

This subsection presents a set of scenarios that were designed with the aim of evaluating the main characteristics of the AMBLE model presented in this paper.

10.2.1 Scenario A This scenario describes the ideal conditions for the model. As mentioned above, the node N receives the T execution request. The node N has full awareness of interaction with the rest of the nodes making up the grid, and therefore this node launches a load balancing operation to carry out the task execution taking into account all the nodes comprising the grid.

For each of the four tests carried out, we measured the execution time in each of the types of nodes (A, B and C)

Downloaded from hpc.sagepub.com by guest on February 25, 2013

Table 2
Scenario A: Improvements and speed up related to the types of nodes (s).

Avg (seg) Amble	Improv. A nodes	Improv. B nodes	Improv. C nodes	Speed up A nodes	Speed up B nodes	Speed up C nodes
5.9	–0.95	–2.04	–2.42	0.84	0.65	0.59
6.964	17.06	12.24	10.84	3.45	2.76	2.56
8.233	38.77	30.49	26.74	5.71	4.70	4.25
17.693	215.24	174.60	157.34	13.17	10.87	9.89

Table 3
Scenario A: Communication overhead and heterogeneous efficiency.

Work load	Grid total comp. power	Grid optimal response time	Comm. overhead	Heter. efficiency
1000	4037.14	0.25	5.65	0.04
5000	4163.20	1.20	5.76	0.17
10000	4255.05	2.35	5.88	0.29
50000	4246.28	11.78	5.92	0.67

Table 4
Scenario B: Improvements and speed up related to the types of nodes (s).

Avg (seg) Amble	Improv A nodes	Improv. B nodes	Improv. C nodes	Speed up A nodes	Speed up B nodes	Speed up C nodes
6.54	–1.59	–2.68	–3.06	0.76	0.59	0.53
7.45	16.57	11.75	10.35	3.22	2.58	2.39
9.906	37.10	28.81	25.06	4.74	3.91	3.53
19.289	213.65	173.01	155.74	12.08	9.97	9.07

Table 5
Scenario B: Communication overhead and heterogeneous efficiency.

Work load	Grid total comp. power	Grid optimal response time	Comm. overhead	Heter. efficiency
1000	4037.14	0.25	10.68	0.03
5000	4163.20	1.20	10.88	0.10
10000	4255.05	2.35	11.78	0.17
50000	4246.28	11.78	12.52	0.49

as well as the total response time of the system using the AMBLE implementation. Tables 2 and 3 present the results obtained in seconds.

10.2.2 Scenario B This scenario raises the non-ideal situation in which all the nodes in the grid are being infra-utilized and therefore they could receive more processes to be executed, but they are located in different auras. The grid client requests the execution of one of the tasks in node N. This node has 10 more nodes inside its aura with a distance equal to 1 and the nine remaining in another aura with a distance equal to 2. Tables 4 and 5 present the improvements, the speed up related to the types of nodes, the communication overhead and the heterogeneous efficiency for the different types of nodes.

Downloaded from hpc.sagepub.com by guest on February 25, 2013

Table 6
Scenario C: Improvements and speed up related to the types of nodes (s).

Avg (seg) Amble	Improv. A nodes	Improv. B nodes	Improv. C nodes	Speed up A nodes	Speed up B nodes	Speed up C nodes
10.924	–5.97	–7.07	–7.44	0.45	0.35	0.32
12.082	11.94	7.12	5.72	1.99	1.59	1.47
14.13	32.87	24.59	20.84	3.33	2.74	2.47
24.29	208.64	168.01	150.74	9.59	7.92	7.21

Table 7
Scenario C: Communication overhead and heterogeneous efficiency.

Work load	Grid total comp. power	Grid optimal response rime	Comm. overhead	Heter. efficiency
1000	4037.14	0.25	10.68	0.03
5000	4163.20	1.20	10.88	0.10
10000	4255.05	2.35	11.78	0.17
50000	4246.28	11.78	12.52	0.49

10.2.3 Scenario C This same scenario also raises the non-ideal situation in which all the nodes in the grid are being infra-utilized and therefore they could receive more processes to be executed, but they are located in different auras. The grid client requests the execution of one of the tasks in node N. This node has 10 more nodes inside its aura with a distance equal to 1 ($aura_1$) and the nine remaining in another aura with a distance equal to 2 ($aura_2$). However, in this situation half of the nodes that are inside $aura_1$ reject the execution of the processes assigned. Then, the load balancing algorithm increases $aura_2$ and therefore the nine remaining nodes can accept any of the processes that are looking for a location. The task delivery process is performed among the nodes located in $aura_2$. While this distribution is being carried out, some of the nodes that are located inside $aura_1$ change their NimbusState so that they could receive new processes. The system will report this change in their NimbusState and those processes that were not assigned among the nodes located in $aura_2$, will be assigned, through the load balancing algorithm, among all those nodes which changed their NimbusState in $aura_1$ and have, therefore, the capacity to receive new processes. Tables 6 and 7 present the improvements, the speed up related to the types of nodes, the communication overhead and the heterogeneous efficiency for the different types of nodes.

10.3 WE-AMBLE: Task Delivery Capability

The following experiment determines WE-AMBLE's capability to deliver tasks and processes. With this purpose we have raised a new scenario requiring the execution of 25 tasks, each of them with 25 processes. Every 20 seconds a group of tasks is sent to the same node of the grid. Figure 6 shows the number of processes executed in each and every node of the grid following WE-AMBLE's task delivery assignment. Nodes with identifiers 1–8 are type A, those with identifiers 9–19 are type B and node 20 is type C. Figure 6 shows that the more powerful nodes execute more processes.

Figure 7 shows the response time of each of the nodes in the grid. Even though each of the nodes executes a different number of processes, all of them finalize at the same time.

From the experimental results shown in Figures 6 and 7), it is possible to conclude that the task assignment made

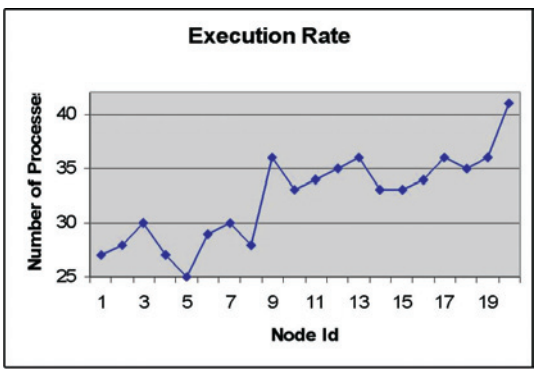

Fig. 6 Number of processes vs. node.

Downloaded from hpc.sagepub.com by guest on February 25, 2013

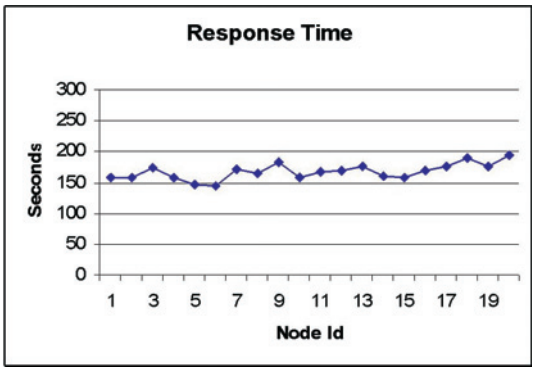

Fig. 7 Response time vs. node.

by WE-AMBLE is proportional to the computational power of each of the nodes, and therefore it uses the grid resource in a optimal way.

10.4 Analysis of Results

Generally speaking, it would be possible to conclude that the experimental results obtained in the tests described above are very successful and corroborate the usefulness of the AMBLE model as applied to workload balancing operations in real heterogeneous grid environments. Tables 2, 4 and 6 show that the performance improvements obtained using this model are excellent in all the scenarios and in almost all the tests. It is notable that the model achieved the worst results in Test 1. These results are a consequence of the small size of the task to be executed, which incurs a considerable communication overhead in this delivery operation, increasing the response times and making these times higher than the one associated with the local execution. As a result of this, the speedup of these experiments is less than 1. An important conclusion can be drawn from these results: if the processes to be processed have a response time lower than a specific threshold (given by the model communication overhead), it would be preferable to carry out a local execution of the task instead of distributing the processes across the grid.

The results for the other scenarios and tests show that the response times exhibit an important improvement not just for the speedup but also for the heterogeneous efficiency and, in general, it would be possible to conclude that, the larger the size of the problem to be solved the better the results achieved. The communication overhead is the factor limiting the performance increase. This overhead is independent of the problem size (as indicated in Tables 3, 5 and 7). In this way, when the problem size increases, the parallelizable portion of the task also increases and therefore the speedup shows a considerable improvement.

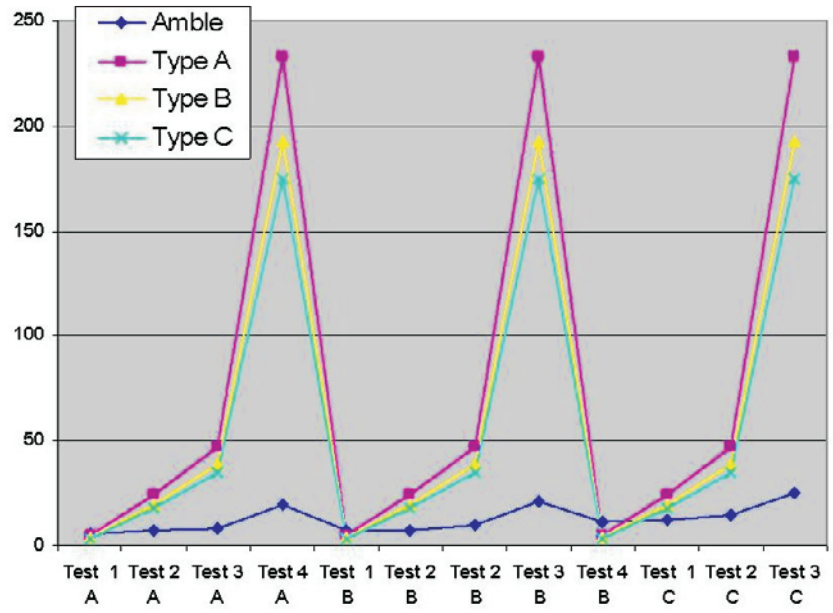

Fig. 8 Response times for the different scenarios and tests.

Downloaded from hpc.sagepub.com by guest on February 25, 2013

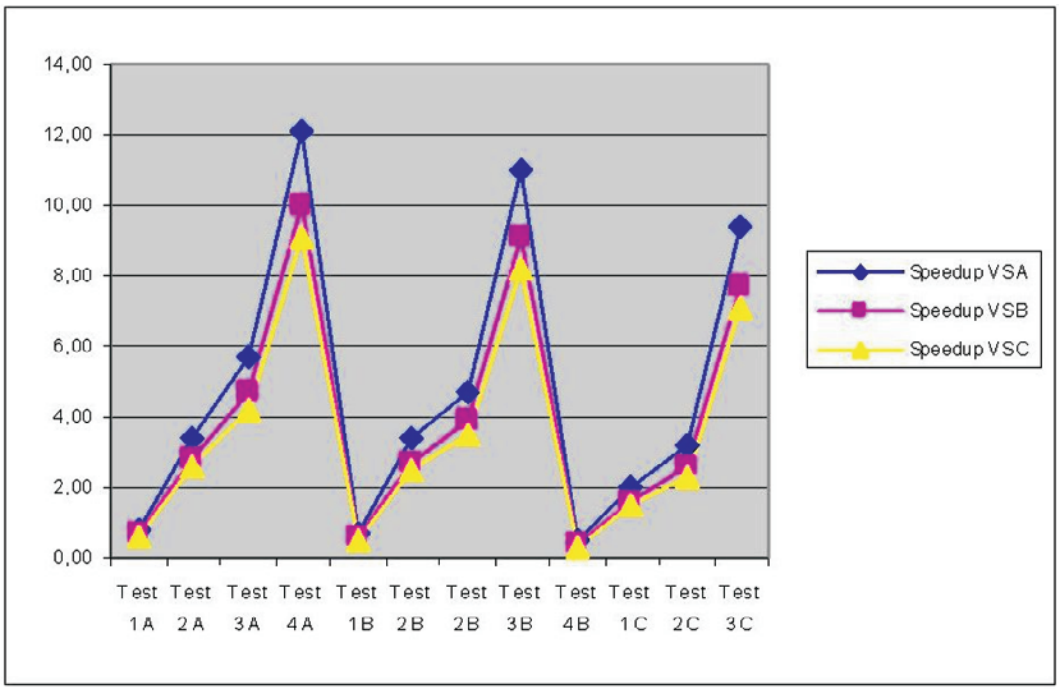

Fig. 9 Speed up for the different scenarios and tests.

Of the scenarios presented in this paper, scenario A gets the best speedup results for the AMBLE model speedup in relation to the local execution. This is a consequence of the ideal conditions, for the execution of the AMBLE model, in which this scenario takes place. Scenario B presents an increment in the aura. This implies that the load-balancing model will require a set of messages to carry out the delivery operation (see Section 7). This communication overhead (see Tables 3, 5, 7) is reflected in the speedup results, but in spite of this communication overhead the results are pretty good.

Scenario C introduced an additional handicap: modifications of the NimbusState of some of the nodes, increasing, even more, the number of messages to be exchanged among the nodes to carry out the delivery operation. However, in spite of this additional communication overhead the results are still very successful. These conclusions are illustrated in Figures 8 and 9.

11 Conclusions

Equilibrating the amount of work assigned to each of the nodes in a grid environment is a complex problem, even more so than for other kinds of parallel systems. This paper presents an awareness model, AMBLE, for balancing the load in collaborative grid environments and collaborative multi-agent systems. This model extends and reinterprets some of the key concepts of the most successful models of awareness in Computer Supported Cooperative Work (CSCW), called the Spatial Model of Interaction (SMI). This reinterpretation merges research carried out in CSCW, agents and grid communities, to create a collaborative and cooperative agent-based grid environment. AMBLE manages the interaction in the environment facilitating the autonomous, efficient and independent task allocation in the environment.

WE-AMBLE, a workflow engine to manage awareness in collaborative grid environments through the AMBLE concepts, has been designed, from the beginning to be a parametrical, generic, open, model that could be extended and adapted easily to new ideas and purposes. This model allows management not just of resources and information but also of interaction and awareness. WE-AMBLE allows: i) control of the user interaction (through the aura concept); ii) guiding the awareness towards specific users and resources (through both focus and nimbus concepts); iii) scaling interaction through the awareness concept.

WE-AMBLE also has the ability of being: i) extended and adapted to new modifications in the model; ii) scala-

Downloaded from hpc.sagepub.com by guest on February 25, 2013

ble to different configurations; iii) re-used to manage different levels of awareness in different infrastructures and virtual organizations; and iv) free of bottlenecks because of its distributed nature.

This model has been evaluated in a real and heterogeneous grid infrastructure. Different scenarios were designed for this purpose. Each of these scenarios was also composed of a set of different computation-intensive tests based in iterations over the Linpack benchmark. These scenarios were designed in such way that each of them introduced some additional handicaps to the previous one. However, in spite of this gradual chain of handicaps, the results obtained in this evaluation have been pretty good.

The most important conclusions that could be extracted from the experimental results presented in this paper are: Firstly, the AMBLE model can contribute to the performance of heterogeneous systems by distributing the workload in an equilibrating way among all the nodes composing the grid; Secondly, the communication overhead in a grid environment is a factor to be considered because of the remarkable limitations in the performance improvements. This overhead does not depend on the problem size, it mainly depends on the dynamism of the grid system, at each and every moment, and therefore it cannot be predicted beforehand. Finally, it is important to highlight that the size of the processes, to be distributed in the grid, has a fundamental impact in the global performance of the system. Those processes with low response times are not suitable for distribution to the grid because the communication overhead could be higher than the local response time, resulting in a worsening of the system.

Acknowledgments

This work has been partially funded by the Spanish Ministry of Education and Science (grant TIC2003-08933-C02) and Government of the Community of Madrid (grants GR/SAL/0940/2004 and S-0505/DPI/0235).

Author Biographies

Pilar Herrero is an assistant professor at the Universidad Politécnica de Madrid, in Spain. She gained her Ph.D. in computer science and was awarded the Extraordinary Prize of Doctorate (Extraordinary Ph.D. Award). Her research interests include awareness management, agents and multi-agent systems and grid computing. In the last few years, she has also been involved in the organization of numerous international events, such as journal publications, conferences and workshops, and since 2005 she has been involved in the organization board of the On The Move (OTM) Federated Conferences as Workshops General Chair. She has been editor of some special issues on the Future Generation Computer Systems (FGCS) and Multi-agent and Grid Systems (MAGS) and several proceedings books. Pilar Herrero has more than 60 publications, some of them in international journals such as IEEE Transactions on Education, Journal of Autonomous Agents and Multi-Agent Systems, Future Generation Computer Systems, International Journal of High Performance Computing Applications, Springer and Presence.

Jose L. Bosque graduated in computer science and engineering from Universidad Politécnica de Madrid in 1994. He received his Ph.D. degree in computer science and engineering in 2003 and the Extraordinary Ph.D Award from the same University. His Ph.D. was centered on theoretical models and algorithms for heterogeneous clusters. He has been an associate professor at the Universidad Rey Juan Carlos in Madrid, Spain, since 1998 and from 2006 he has been associate professor at the Universidad de Cantabria. His research is in the area of computer architecture, with special emphasis on high performance computers: parallel and distributed processing, parallel computational models, performance and scalability evaluation and load balancing algorithms. He has published more than 60 papers in international journals and conferences, such as IEEE Transactions on Parallel and Distributed Systems, Journal of Parallel and Distributed Computing, Multi-agents and Grid Systems and Journal of High Performance Computing Applications. In the area of his research he has participated in 22 European and Spanish projects and 12 projects with European companies. He has directed a Ph.D. on load-balancing algorithms in heterogeneous clusters.

Manuel Salvadores has an M.S. in computer science and is a Ph.D. student at Universidad Politécnica de Madrid, in Spain. His research includes distributed transactions, service awareness and grid computing. Due to his Ph.D. he has collaborated with National Science Centre in the OGSA-DAI, researching in the area of distributed transactions for grid computing environments. Also he has published several papers based on DCP-Grid, a distributed commit protocol for grid computing, based on Web Services and grid computing standards. His last papers are focused on service awareness for grid computing and his future research will be merge the service awareness research with his background on distributed transactions for grid computing. In the enterprise area, during the last few years, he has been leading the construction of Intelligent Mainframe Grid as Software Architect. Intelligent Mainframe Grid is a grid computing framework oriented to financial business. This product is being adopted by one of the most important banks in Spain in order to download batch processes from a centralized system.

Downloaded from hpc.sagepub.com by guest on February 25, 2013

María S. Pérez received her M.S. degree in computer science in 1998 from the Universidad Politécnica de Madrid, her Ph.D. degree in computer science in 2003 and the Extraordinary Ph.D Award at the same university. From 1998 she has been an associate professor at the Universidad Politécnica de Madrid. Her research interests include high-performance and grid computing, parallel I/O, autonomic computing and data mining. She is coauthor of four books, four book chapters and she has published more than 60 articles in journals and conferences, some of them in international journals such as IEEE Transactions on Education, Future Generation Computer Systems, Journal of Parallel and Distributed Computing and International Journal of High Performance Computing Applications. She has been involved in the organization of several grid related workshops and conferences, and she has edited several proceedings books and special issues. Currently she is involved in the Ontogrid European project (FP6-511513), whose main goal is to provide a reference semantic grid architecture. She is also member of the editorial board of the Journal of Autonomic and Trusted Computing.

Notes

1 WS-Security: http://www-106.ibm.com/developerworks/library/ws-secmap accessed February 2007

2 World Wide Web Consortium, http://www.w3.org/ accessed February, 2007;
Web Services Activity, http://www.w3.org/2002/ws/ accessed February, 2007

3 SOAP Specifications, http://www.w3.org/TR/soap/ accessed February, 2007

4 WS-Addressing, http://www.w3.org/Submission/ws-addressing/ accessed February, 2007;
WSN, Web Service Notification, www.oasis-open.org/committees/wsn/ accessed February, 2007;
WS-Topics, docs.oasis-open.org/wsn/2004/06/wsn-WS-Topics-1.2-draft-01.pdf accessed February, 2007;
WSRF. www.oasis-open.org/committees/wsrf/ accessed February, 2007

References

Beltrán M., Bosque, J. L., and Guzmán, A. (2004). Resource dissemination policies on grids, in *On the Move to Meaningful Internet Systems*, Lectures Notes in Computer Science, Berlin: Springer-Verlag, pp. 135–144.

Benford S. D. and Fahlén, L. E. (1993). A spatial model of interaction in large virtual environments, in *Proceedings of the Third European Conference on Computer Supported Cooperative Work (ECSCW'93)*, Kluwer Academic Publishers, pp. 109–124.

Berman, F., Wolski, R., Figueira, S., Schopf, J., and Shao, G. (1996). Application-level scheduling on distributed heterogeneous networks, in *Proceedings of the ACM/IEEE Conference on Supercomputing (SC'96)*, Los Alamitos, CA: IEEE Computer Society.

Chen, J. and Yang, Y. (2006a). Multiple states based temporal consistency for dynamic verification of fixed-time constraints in grid workflow systems, *Concurrency and Computation: Practice and Experience.*

Chen J. and Yang, Y. (2006b). Selecting necessary and sufficient checkpoints for dynamic verification of fixed-time constraints in grid workflow systems, in *Proceedings of the 4th International Conference on Business Process Management (BPM2006)*, Lectures Notes in Computer Science 4102, Berlin: Springer-Verlag, pp. 445–450.

Cheung W. K. et al. (2004). Dynamic resource selection for service composition in the grid. *IEEE/WIC/ACM International Conference on Web Intelligence*, Los Alamitos, CA: IEEE Computer Society Press, pp. 412–418.

Das, S. K., Harvey, D. J., and Biswas, R. (2001). Parallel processing of adaptive meshes with load balancing. *IEEE Trans. on Parallel and Distributed Systems*, **12**: 1269–1280.

Deelman, E., Blythe, J., Gil, Y., and Kesselman, C. (2004). Workflow management in GriPhyN, in *Grid Resource Management: state of the art and future trends,* Kluwer Academic Publisher, pp. 99–116.

Fahlén, L. E. and Brown, C. G. (1992). The use of a 3D Aura metaphor for computer based conferencing and teleworking, in *Proceedings of the 4th Multi-G Workshop*, pp. 69–74.

Foster, I., Jennings, N. R., and Kesselman, C. (2004). Brain meets brawn: Why grid and agents need each other, in *Proceedings of the 3rd International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS 2004)*.

Foster, I., Kesselman, C., Nick, J., and Tuecke, S. (2002). The Physiology of the grid: An open grid services architecture for distributed systems integration, *Open Grid Service Infrastructure WG,* Global Grid Forum.

Greenhalgh, C. (1999). *Large Scale Collaborative Virtual Environments*, PhD thesis, University of Nottingham. Berlin: Springer-Verlag.

Greenhalgh, C. M., and Benford, S. D. (1995). MASSIVE: A virtual reality system for tele-conferencing, *ACM Transactions on Computer Human Interfaces*, **2**(3): 239–261.

Harchol-Balter, M. (2001). Job placement with unknown duration and no preemption, *ACM SIGMETRICS Performance Evaluation Review*, **28**(4): 3–5.

Harchol-Balter, M. and Downey, A. B. (1997). Exploiting process lifetime distributions for dynamic load balancing, *ACM Trans. on Computer Systems*, **15**(3): 253–285.

Kunz, T. (1991). The influence of different workload descriptions on a heuristic load balancing scheme, *IEEE Transactions on Software Engineering*, **17**(7): 725–730.

Letsch, T. (2000). *Redesign and Implementation of a Mobile Agent System Compliant with the MAFFinder Part of the MASIF Standard*, PhD thesis, Department of Informatics, Technischen Universität München.

Oinn, T. et al. (2004). Taverna: a tool for the composition and enactment of bioinformatics workflows, *Bioinformatics*, **20**(17): 3045–3054.

Downloaded from hpc.sagepub.com by guest on February 25, 2013

Pastor L. and Bosque, J. L. (2001). An efficiency and scalability model for heterogeneous cluster, in *Proceedings of the 3rd IEEE International Conference on Cluster Computing*, Los Alamitos, CA: IEEE Computer Society Press, pp. 427–434.

Russ, S. H., Robinson, J., Flachs, B. K., and Heckel, B. (1998). The Hector distributed runtime environment, *IEEE Transactions on Parallel and Distributed Systems*, **9**(11): 1002–1114.

Salvadores, M., Herrero, P., Pérez, M. S., and Sanchez, A. (2005). Modelling the N + ROWA model approach inside the WS-Replication resource, in *Second International On the Move Federated Conferences (OTM'05)*, Lecture Notes in Computer Science 3762, Berlin: Springer, pp. 397–405.

Shen, W. et al. (2002). Adaptive negotiation for agent-based grid computing, *Journal of the American Statistical Association*.

Tannenbaum, T., Wright, D., Miller, K., and Livny, M. (2002). Condor- A Distributed Job Scheduler, in *Beowulf Cluster Computing with Linux,* Cambridge, MA: MIT Press.

University of Tennessee (2006). Performance of various computers using standard Linear equations software, Technical Report CS-89-85, University of Tennessee, http://www.netlib.org/benchmark/performance.pdf accessed February, 2007.

van Nieuwpoort, R. V., Wrzesinska, G., Maassen, J., Kielmann, T., and Bal, H. E. (2006). Adaptive load balancing for divide-and-conquer grid applications, *Journal of Supercomputing*.

von Laszewski, G. et al. (2004). GridAnt: A client-controllable grid workflow system, in *Proceedings of the 37th Annual Hawaii International Conference on System Sciences (HICSS'04)*, Los Alamitos, CA: IEEE Computer Society Press.

Xiao, L., Chen, S., and Zhang, X. (2002). Dynamic cluster resource allocations for jobs with known and unknown memory demands*, IEEE Trans. on Parallel and Distributed Systems*, **13**(3): 223–240.

Xu, C. and Lau, F. (1997). *Load balancing in parallel computers: theory and practice*. Kluwer Academic Publishers.

Yu J. and Buyya, R. (2005). A taxonomy of scientific workflow systems for grid computing, *ACM SIGMOD Record*, **34**(3): 44–49.

Zhang, Z. and Luo, S. (2003). Constructing grid system with mobile multiagent. *Proc. of the 2nd Int. Conference on Machine Learning and Cybernetics, Xi'an*.

Zomaya, A. Y. and Teh, Y.-H. (2001). Observations on using genetic algorithms for dynamic load-balancing, *IEEE Trans. on Parallel and Distributed Systems*, **12**(9): 899–911.

Downloaded from hpc.sagepub.com by guest on February 25, 2013