Agent-based Grid Resource Management

A. Lenica[1], F. Ogel[1], F. Peshanski[2], J.-P. Briot[2]

[1] France Telecom - R&D/MAPS/AMS,
38-40 rue du general Leclerc 92794 Issy les Moulineaux, France
firstname.name@francetelecom.com
[2] Laboratoire d'Informatique Paris VI,Université Pierre et Marie Curie,
4, place Jussieu, 75252 Paris Cedex 05, France
firstname.name@lip6.fr

Abstract. As grid computing becomes a de facto standard for hosting calcul-intensive applications, current grid middleware technologies do not deal with interactive (i.e. non-batch) applications. From a Telco (or any service-provider) perspective, such class of applications is highly sensitive. We propose a grid service platform that dynamically provisions resources for both interactive and batch applications to meet their QoS constraints while ensuring good resource mutualization. Moreover, we believe that relying on an agent-based approach for resource management will allow a more flexible, robust and scalable solution.

1 Introduction

Traditionaly, business and multimedia applications are deployed on top of isolated clusters, statically over-provisionned to handle peak loads. As a consequence, resource utilization rates within such infrastructures are typically low (most often between 20% and 30%). Pooling resources together to form a grid service-platform seems a natural way to enhance resource utilization as well as to reduce costs. However, even though grid computing becomes a widely spread solution in support of calcul-intensive applications, both within the academic and industrial worlds, non-batch applications (interactive, multimedia, etc.) are not addressed. Actual grid middleware technologies do not deal with applications such as VoIP or visioconference. Hence, the need for a grid service-platform aimed at hosting highly heterogeneous applications, ranging from multimedia servers (such as VoIP or videoconfernce) to more traditionnal calcul-intensive applications (such as magnetic field simulations and codecs optimization), and data-oriented applications (such as billing or datamining for fraud detection and churn analysis). Whereas resource allocation can be delayed or statically performed in case of batch applications, interactive ones exhibit dynamic and stringent QoS requirements on a possibly long run. Hence, their load is variable over time, requiring dynamic resource allocation. To enable the cohabitation of both classes of applications while satisfying QoS requirements during the complete lifecycle of each application requires from a grid platform: (i) to perform service differentiation; (ii) to perform dynamic allocation; (iii) to adapt resource allocation policies; (iv) while maintaining a good resource utilization rate. Besides, in a grid world, this arbitrator would be a central component, like any resource broker. However for scalability and dependability issues, a decentralized arbitrator would exhibit better properties. We join [8] in the belief that the specificities of an interactive grid are best matched by a multi-agent based approach. To supply our grid resource allocator with the necessary flexibility, scalability and dependability, we propose in this paper a distributed agent-based abstract model. We consider the various allocation semantics as agent behaviours and rely on negociation between agents to achieve matchmaking.

The remainder of this paper starts by presenting a case study based on existing applications in Section 2. Section 3 describes our architecture with respect to the previous case study. Related works appear in Section 4, followed by conclusions and perspectives in Section 5.

2 Case study

To illustrate our proposition, we use a case study based on existing applications: a videoconference service, a data-oriented fraud detection application and a more calcul-intensive videocodecs optimization/benchmark tool.

Our goal is to design an architecture that will allow such heterogeneous applications to coexist on a single platform with respect to their QoS and performance needs. Every available resource should be used and most *important*, services should be able to preempt resources from less important ones. The *importance* of a service corresponds to a *business value*, which can be defined using various utility functions: based on the service contribution to the global benefits, on client class, their workload, etc. In our case, we consider static business values representing class of importance with respect to business activities: the visioconference service is more important than internal batch applications.

2.1 Resources

Physical resources in a Telco grid may include clusters of workstations and servers geographically distributed over long distances, with arbitrary architectures, operating systems and libraries. Each resource is described by a resource profile, providing the necessary semantic markup (architecture, physical resources such as RAM or storage, operating systems, etc.) to manage it. Resources might be virtual resources multiplexed on top of physical resources, as with Xen-like [6] virtualization tools. Although on-going work is also done on the mapping of virtual resources to physical resources, we do not present it here.

2.2 Hosted applications and services

In this paper, we do not distinguish between the terms application and service and therefore we use them interchangeably. Since we consider legacy applications, external controllers have to be specifically developped to interface with the underlying platform. Application controllers encapsulate some application-specific features such as QoS monitoring and associated resource requesting, as well as application components' setup and configuration.

We gather batch applications behind a *Batch Queue Service* that deploys them for execution upon a pool of known resources. Its associated controller has a greedy behaviour: it claims each idle resource available for executing batch jobs. The Batch Queue Service corresponds to the traditional entry point of a data/calcul oriented grid that manages its own resources to handle submitted workflows. We currently use Condor [17] for such purpose.

The visioconference service represents a legacy application, originally written to be deployed on a dedicated cluster. It balances visioconferences amongst a set of pre-existing conference bridges, hence it needs a *controller* to: (i) monitor the QoS associated with the service and some specific performance/load metrics (typically the number of active videoconferences); (ii) interact with the underlying platform, and in particular negociate additional resources when needed.

3 Architecture for dynamic resources provisionning

Most grid middleware technologies are explicitly designed to deal with batch-oriented applications and thus implicitly use this hypothesis for resource management. Adversely, supporting heterogeneous applications requires having an application-neutral resource manager. Thus we propose to provide a dynamic resource allocator that provisions resources to requesting services. However, dealing with heterogeneous applications implies heterogeneous QoS requirements and *business values*. Consequently, we introduce service differentiation concerning resource management to arbitrate amongst requests. Available resources are allocated to most *valued* requests and resources used by low-value services can be preempted for dynamic re-allocation to higher-value applications. Moreover, hosting heterogeneous applications requires supporting different allocation semantics. Whereas, a Batch Queue Service simply requests any resource matching a given profile, more constrained services can exhibit additional requirements, such as timeouts (*i.e.* a delay after which the resource is not useful anymore), or different semantics, such as atomic co-allocation (*i.e.* allocate a whole set of resources matching some given criteria or none).

To capture the heterogeneity of allocation semantics, we adopt a heterogeneous society of software agents, each allocation policy being implemented as an agent behaviour. To add a novel policy to the platform boils down to enrich it with a new class of agents, without having the need to modify the existing ones. Moreover, ensuring good resource management starts by ensuring a good dependability of the resource allocator, so that resources remain available when faults occur. Hence the use of a distributed set of cooperating *agents* for resource management. Using cooperative distributed agents also helps with scalability and reactiveness issues in such potentially large scale distributed environments.

3.1 A generic architecture for resource management

Our architecture deals with resource-centered decision-making. To achieve this purpose, we propose an agent-based resource manager supporting arbitrary resource allocation semantics. The following figure 1 gives an overview of the general agent-based architecture. It consists of four layers: (i) an applicative layer; (ii) associated application controllers responsible for maintaining all metadata associated to the application, monitoring its performance (through QoS metrics) and issuying the requests for the adequate resources when needed; (iii) an infrastructure layer consisting of virtualized grid resources; (iv) the core layer of resource management agents.

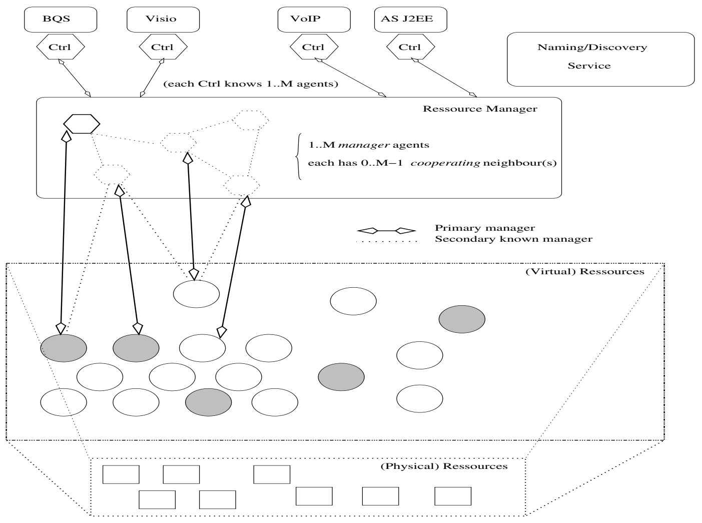

Fig. 1. A generic resource manager.

Application controllers use a Naming/Discovery service to find one or more resource management agents that advertise desired semantics and properties (atomic co-allocation, timed-out allocation, etc.). They send requests (resource profiles and possible QoS constraints) to those agents. A resource profile typically describes both hardware and software informations (such as architecture, operating system, available memory and storage, etc.). Requests also contain additional metadata, such as an allocation mode (agents can support multiple allocation semantics) and some semantic-specific metadata, such as a time-out if any or a *locality*[1] constraint on requested resources. We do not directly address consistency issues emerging from parallel requests between multiple resource

[1] Locality can be expressed in terms of network latency or, at a coarser grain, as a subnetmask.

management agents: controllers are responsible for dealing with it, for example by sending requests sequentially or split them into smaller independant requests that agents can handle in parallel.

Each resource interact with the platform, and in particular with the resource management agents, through a local agent. Resources are dynamically *priced* with a business value. We use a simple model: idle resources have a null value, while allocated resources inherit from the business value of the application they were allocated to. Hence, a resource is considered to be available for any application having a strictly higher (business) value. As a consequence, idle resources are available to anyone. This model will be further extended to differentiate resources: for example, powerful resources and very specific (rare) profiles could be given higher values than common resources. Towards our first prototype, we rely on application controllers to release unused resources. However, we plan to use resource local agents to implement a garbage collecting mechanism: allocated but unused resources use a timeout to trigger an alarm toward their primary manager agent, which sends in turn a *release* message to the corresponding controller. The resource is then preempted and put back to an available/idle state.

Considering the mapping between resources and their managers, having a strict *one-to-many* mapping (*i.e.* one manager dedicated to an exclusive set of resources, brings dependability issues: when the manager fails, associated resources are not available anymore. On the other hand, having multiple agents managing shared resources typically raises consistency issues. We use a simple *replication* schema: any resource has a single primary manager, but is also known by some of its manager neighbours. Those neighbours are also known by the resource and will act as secondary managers, in case of failure of primary. In such case, we propose to let the resource detect its primary's fault and register itself to a known neighbour.

Resource management agents match requests to available resources (*Match-Making*) and arbitrate between requests with respect to Business Values (*service differentiation*). Each agent implements one or more allocation semantics, and advertises its capabilities to the Naming/Discovery Service. Their behaviour depends on the allocation semantic they implement and extends the basic behaviour consisting in matching a requested resource profile (including a business value) with an available resource (*i.e.* idle or allocated to a lesser-valued application) within its set of primary resources. Upon failure, the agent behaves as an application controller to negociate resources from a neighbour. Based on its knowledge about its neighbours and their potential resources, an agent can determine the most suitable neighbour to handle the request. It then uses a delegation scheme so that the request can be routed through agents toward completion (or expiration). Delegated requests inherit the business value associated with the application controller they originated from. Metadata management between agents is similar to cooperation protocol between Web caches, such as Relais [12] or Summary Cache [7]. This basic behaviour is common to all the management agents, enabling them to "understand" each other and cooperate during this phase of matchmaking. Agents further specialize according to different allocation policies. For example, co-allocation requires further cooperation between agents to perform end-to-end resource reservation.

3.2 A simple example

In order to illustrate how resource arbitration works, we consider the applications presented in section 2.

We introduce service differentiation by associating priorities (or *Business Values*) to services. Batch applications are typically considered less important than interactive applications: in our case the visioconference service is considered to be of higher importance from a business perspective. The Batch Queue Service has a greedy resource consumption: it claims every available resources to execute its jobs. The visioconference *only* needs resources to maintain QoS and thus asks for more when its load (defined as the number of visioconferences per bridge) crosses a predefined threshold.

Applications deployment and configuration is handled by application controllers: they find a resource manager and ask for resources to setup a running service. Once up and running, the Batch Queue Service controller asks for resources as long as it has jobs waiting for execution: it simply tries to form a resource pool large enough to handle all its jobs in parallel. The controller of the videoconference service simply monitors conference load and sends a request containing a description of a resource needed to host a videoconference server. Thus it tries to anticipate peak loads to prevent QoS perturbations or user requests rejection. In this configuration, the Batch Queue System uses most of idle resources, until the videoconference-service load increases.

The above allocation semantics may translate to the following agent behaviours: when requested by the controller of the videoconference service, resource manager agents will start preempting resources from the Batch Queue Service.

As a result, on one hand Batch Queue Service will only get idle resources, on the other hand videoconference service gets both idle and preempted resources. At the end, the videoconference service gets as much resource as it needs (if enough) and, due to its greedy behaviour, the Batch Queue Service makes use of every remaining resource.

Introducing a third service, for example a J2EE application server, having an intermediate business value, results in a similar hierarchy: the most valuable services gets all the resources they need, with respect to their priority and the Batch Queue Service, which represents internal low-priority applications in our case-study, exploits the remaining idle resources. Hence we preserve application priorities (and QoS constraints) while ensuring a good resource rationalization.

4 Related works

As stated in the introduction, resource management systems explicitly targeted at interactive grids are scarce. Recent projects include [11], which proposes an abstract agent-based architecture to enforce SLAs during application runtime. Compared to the work presented here, dynamic allocation is not adressed: the QoS requirements are supposed static and the only enforcement policies envisaged are prioritization mechanisms (or application termination), which are not compatible with the interests of a telecommmunication operator. Besides, most integrated management systems in grids exhibit static resource management services, incompatible with our purposes. In most cases, jobs are submitted and exit without the possibility of reallocation in course of execution, as in **Globus** [9], **Sun's Grid Engine** or **Ninf** [13]. As for **Condor** [17], if a participating node is overloaded or becomes unavailable, the Condor daemon takes a snapshot of the job and resumes it on another node. But due to its snapshot algorithm, Condor is suitable only for batch jobs. In **Legion** [14], neither the scheduler nor the scheduling policies are mandated. The framework provides mechanisms to construct ad-hoc schedulers and permits resource specification directly for a run. But the general scheduling architecture supports only one-shot negociations between client and provider and therefore, no dynamic reallocation (same for **Nimrod/G** [2]. Other constraints are the noninterference with the applicative code, which is violated by projects like **NetSolve** [16] (where applications must use one of the APIs provided by the system to perform RPC-like computations) or the automation of administrative tasks, which is not found in **AppLeS** [1] (no tool level scheduling solution provided but instead, an ad-hoc scheduler agent must be developped for each application).

Few projects investigate the dynamic resource allocation on grids. Amidst of them, CoordAgent [10], [15] and ARMS [3]. **CoordAgent** proposes an agent-based solution where mobile agents represent user job requests. The agent encapsulates the code, searches the grid for adequate resources, launches the job, and migrates it to another node in case the current host becomes unavailable. Although the system uses migration for fault-tolerance purposes, it could indeed be used to support dynamic resource allocation for interactive processes, given the possibility to plug-in user/administration policies. In regard to our objectives, the project shortcomings of CoordAgent lie in that it requires modification of the applicative code in order to insert checkpoints and mandates Java or C++. Without alteration of the user code, [15] also proposes to enhance the resource broker by providing it with migration capabilities via reflective techniques. The broker gathers dynamic information about the state of the resource during runtime and reports it to a monitor which in turn computes predictive information. This predictive information is used by an Adapter Manager to make a decision whether job migration is required. The principal drawback to this proposal is the obligation for the applications to be developped in OpenJava, and thus its inability to handle legacy applications. The last project presented here is ARMS. **ARMS** tackles the issues of scalability and adaptability in resource management systems though the use of reconfigurable agents. Each agent represents a grid resource which it manages locally, and cooperates with the other agents to enable resource allocation by exchanging with them service advertisements and discovery requests. A special agent possesses a global view of the system and is capable of modeling and simulating the peer agent's performances during runtime, while optimizing its behaviour (e.g. service advertisement frequency). The system is both scalable (the agents compose a homogeneous society except for the special agent, which does not constitue a single point of failure) and adaptive (through the use of PACE performance prediction toolkit to compute a tradeoff between complexity of advertisement and complexity of discovery). But the model relies on hypothesises about the application as the attribution of one job per machine, which is relevant for batch applications but not interactive ones, targeted in this work.

5 Conclusion and perspectives

Whereas most grid middlewares focus on batch jobs execution, a grid service platform in a Telco or service-provider context has to deal with both batch and interactive applications. Not only does static resource provisoning lead to poor resource rationalization, but it fails also to support dynamic QoS requirements. To face such heterogeneous and dynamic context, we propose a generic resource manager able to support arbitrary classes of applications, while avoiding systematic over-provisioning of resource and ensuring service differentiation.

This paper presented the architecture of such a resource manager. It is based on a multi-agent approach to capture the heterogeneity of hosted applications (in terms of allocation semantics) as well as to provide good dependability.

We are currently investigating self-organization heuristics to provide this architecture with autonomic capabilities. In particular we would like to handle transverse aspects, such as fault tolerance, through organizational strategies. A first prototype is under development, based on the ProActive Java platform [4, 5].

References

1. F. Berman, R. Wolski, H. Casanova, W. Cirne, H. Dail, S. Figueira, J. Hayes, G. Obertelli, J. Schopf, G. Shao, S. Smallen, S. Spring, A. Su, and D Zagorodnov. Adaptive computing on the grid using apples. *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, 14(4):369–382, 2003.
2. R. Buyya, D. Abramson, and J Giddy. Nimrod/g: An architecture for a resource management and scheduling system in a global computational grid. In USA IEEE Computer Society Press, editor, *Proceedings of the Fourth International Conference on High Performance Computing in Asia-Pacific Region (HPC ASIA'2000)*, China, 2000.
3. J. Cao, S. A. Jarvis, S. Saini, D. J. Kerbyson, and G. R. Nudd. Arms: an agent-based resource management system for grid computing. *Scientific Programming, Special Issue on Grid Computing*, 10:135–148, 2002.
4. Denis Caromel. Toward a method of object-oriented concurrent programming. *Communications of the ACM*, 36(9):90–102, 1993.
5. Denis Caromel, Wilfried Klauser, and Julien Vayssière. Towards seamless computing and metacomputing in Java. *Concurrency: Practice and Experience*, 10(11–13):1043–1061, 1998.

6. B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, I. Pratt, A. Warfield, P. Barham, and R. Neugebauer. Xen and the art of virtualization. In *Proceedings of the ACM Symposium on Operating Systems Principles*, October 2003.
7. Li Fan, Pei Cao, Jussara Almeida, and Andrei Z. Broder. Summary cache: a scalable wide-area Web cache sharing protocol. *IEEE/ACM Transactions on Networking*, 8(3):281–293, 2000.
8. I. Foster, N. Jennings, and C. Kesselman. Brain meets brawn: Why grid and agents need each other, 2004.
9. I. Foster and C. Kesselman. Globus: A metacomputing infrastructure toolkit. *Intl J. Supercomputer Applications*, 11(2):115–128, 1997.
10. M. Fukuda, Y. Tanaka, N. Suzuki, L. F. Bic, and S. Kobayashi. A mobile-agent-based pc grid. In *Autonomic Computing Workshop Fifth Annual International Workshop on Active Middleware Services (AMS'03)*, pages 142–150, 2003.
11. R. Kumar, V. Talwar, and S. Basu. A resource mangement framework for interactive grids. In *Concurrency and Computation: Practive and Experience*, volume 16, pages 489–501. "John Wiley & Sons, Ltd.", 2004.
12. Mesaac Makpangou, Guillaume Pierre, Christian Khoury, and Neilze Dorta. Replicated directory service for weakly consistent replicated caches. In *Proceedings of the 19th IEEE International Conference on Distributed Computing Systems (ICDCS '99)*, May 1999.
13. H. Nakada, Y. Tanaka, S. Matsuoka, and S. Sekiguchi. The design and implementation of a fault-tolerant rpc system: Ninf-c. In *Proceedings of the Seventh International Conference on High Performance Computing and Grid in Asia Pacific Region (HPC Asia 2004)*, pages 9–18, July 2004.
14. A. Natrajan, M. Humphrey, and A. Grimshaw. Capacity and capability computing in legion. International Conference on Computational Science, May 2001.
15. A Othman, P Dew, K Djemame, and I Gourlay. Toward an interactive grid adaptive resource broker. In S J, editor, *Proceedings of the Second UK All Hands e-Science Meeting 2003 (EPSRC'03)*, 2003.
16. K. Seymour, A. Yarkhan, S. Agrawal, and J. Dongarra. Netsolve: Grid enabling scientific computing environments. In *Grid Computig and New Frontiers of High Performance Processing*, volume 14 of *Advances in Parallel Computing*. Elsevier, 2005.
17. T. Tannenbaum, D. Wright, K. Miller, and M. Livny. Condor – a distributed job scheduler. In Thomas Sterling, editor, *Beowulf Cluster Computing with Linux*. MIT Press, October 2001.