Description of a Lightweight Bartering Grid Architecture

Cyril Briquet and Pierre-Arnoul de Marneffe

Department of Electrical Engineering & Computer Science, University of Liège,
Montefiore Institute, B37, B-4000 Liège, Belgium
email: {C.Briquet,PA.deMarneffe}@ulg.ac.be

Abstract

Automated resource exchange and negotiation between participants of a Virtual Organization, or Peers of a Peer-to-Peer Grid, is an important feature of Grid computing because it enables scalable cooperation between entities under separate administrative control. Automated negotiation and accounting of resource consumption have been studied, and market-based resource exchange methods have been proposed. However, there currently exist few simulators of resource exchange accounting or actual Grid middlewares supporting negotiation with automated resource usage accounting between separate entities. We propose a Lightweight Bartering Grid (LBG) architecture suitable to the development of Peer-to-Peer Grids based on bartering (i.e. automated and accounted resource exchange), where Peers model their environment. We present the LBG architecture as well as a simulator and a middleware under development that both instantiate it.

1 Introduction

The convergence between the Grid computing and Peer-to-Peer (P2P) domains [1] leads to consider so-called Peer-to-Peer Grids [2]. In P2P Grids, formation of Virtual Organizations (V.O.) is more often bottom-up (i.e. the V.O. emerges out of the resource sharing without prior definition) and *in-Grid* than in traditional Grids where participants decide *out-of-Grid* to share resources and enter the V.O. in a top-down fashion (i.e. the V.O. is explicitly defined before participants engage in resource sharing) [3].

To allow for sustainable resource exchanges to take place in P2P Grids, Peers should use resource exchange methods that are perceived as equitable by the other Peers. Bartering is a decentralized, non-monetary, market-based resource exchange method [4, 5, 6] that enables automated and accounted resource exchange between Grid Peers. As each Peer takes its own Resource exchange (i.e. consumption, supplying) decisions independently of other Peers, bartering is intrinsically scalable as it does not need centralized management or monitoring.

In this paper, we present an architecture of a Lightweight Bartering Grid (LBG) with the objective to enable fast and easy development of accounting, negotiation, scheduling and queueing algorithms supporting bartering in a P2P Grid. This architecture is closely related to the OurGrid middleware [2], although with more focus on modelling of the environment. The actual imple-

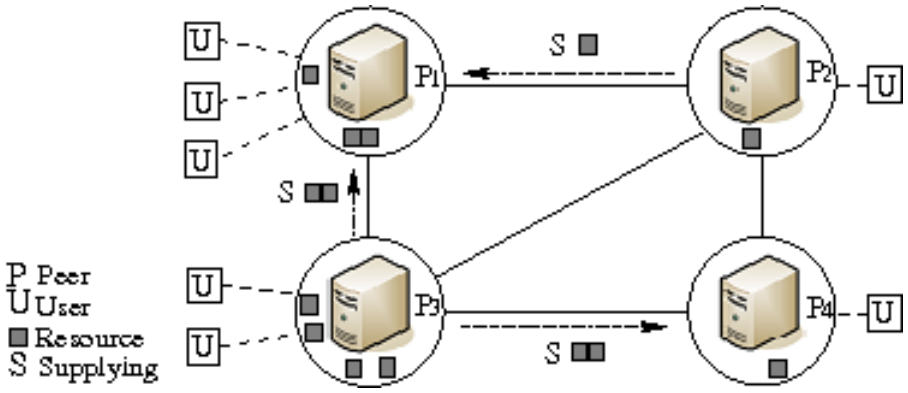
mentation of this architecture has been realized in the Java environment, with an emphasis on openness to new algorithms. Once an algorithm is coded in Java and implements the expected interfaces, it can be simulated with accuracy and then immediately used, without any modification, in a production environment. This is possible because the architecture is easily deployed as a simulator and as a middleware, thanks to code sharing of much of the code base, which is also open to new deployment modes.

The rest of the paper is structured as follows: Section 2 presents the LBG architecture, Section 3 describes the components of a Peer, Section 4 explains how the LBG architecture can be deployed as a simulator and as a middleware, and finally Section 5 summarizes and discusses future work.

2 Architecture of a Lightweight Bartering Grid

2.1 Peer-to-Peer Grid model

In the context of our work, we consider independent entities under separate administrative control (see Figure 1). Each entity is represented by a software agent that is called a *Peer*. We make the hypothesis that a given Peer exists to serve its *Users* but we make no assumption on their prioritizing. The Users of a given Peer submit *Jobs* composed of independent computational *Tasks* (possibly sharing the same input data).



Peer	Resources	Need	Bartering
P_1	3	6	Consumption: 3 own, 1 from P_2, 2 from P_3
P_2	1	0	Supplying: 1 to P_1
P_3	4	0	Supplying: 2 to P_1, 2 to P_4
P_4	1	3	Consumption: 1 own, 2 from P_3

Fig. 1: P2P Grid with synchronized Bartering (at time t)

The goal of each Peer is to serve its *Users*. To this end, each Peer has two

possibilities. Firstly, each Peer usually owns (i.e. controls) several *Resources*. Secondly, each Peer can consume Resources of other Peers, meaning that it executes some of its Tasks on the Resources of other Peers. In the LBG architecture, a *Grid* is a transient network of Peers exchanging Resources that emerges in a bottom-up fashion.

Resource exchange is a way to potentially speed up Jobs completion time. Synchronized consumption by a given Peer of multiple Resources belonging to other Peers (i.e. nontrivial QoS in Foster's well-known 3-point checklist) may indeed dramatically reduce Jobs completion times. In return, the given Peer should help the Resource suppliers by supplying them with its own Resources, preferably when it has no Job to complete for its Users and when the other Peers have Jobs to complete for theirs. It is therefore more advantageous for Peers to group together with Peers having temporally heterogeneous (dissimilar) Resource needs, rather than with Peers having similar needs.

The work most related to the LBG architecture is OurGrid [2, 4]. The main and most important difference is that an LBG User is lighter than an OurGrid user (called *MyGrid*), which also embeds some Peer features such as scheduling. In LBG, each Peer explicitly models its environment, including an estimation of the behaviour and performance of its suppliers. If this modelling were done by Users, there would be no sharing of information regarding the different Tasks submitted to the Peer, and some interference might possibly arise in the scheduling and negotiation processes, leading to degraded performance.

2.2 Peer model

A Peer can be seen as a system which receives and processes computing *Requests* submitted by its Users or by other Peers. A computing Request is a *Job* composed of a set of independent computational *Tasks* (a so-called *Bag of Tasks* [2]) to be completed. Resources are supplied to execute work units of other Peers at the Task level.

The main components of a Peer are a set of thread-safe, loosely coupled managers that take care of the different aspects of operating a Peer (see Figure 2). Each Peer manages incoming Requests and Tasks queues, discovers other Peers, manages information about the state of its Resources and the state of the Resources supplied by other Peers, negotiates Resource exchange with other Peers, schedules and manages the execution of Tasks.

3 Managers of a Peer

3.1 Request Manager

The Request Manager is the Peer component responsible for the queueing of incoming Requests, both from the Peer Users and from other Peers. It is merely a passive container from which Tasks ready to be executed can be extracted.

Two queues are maintained to store Requests following a simple (e.g. FIFO) policy. Support for several classes of priority could be implemented easily, and

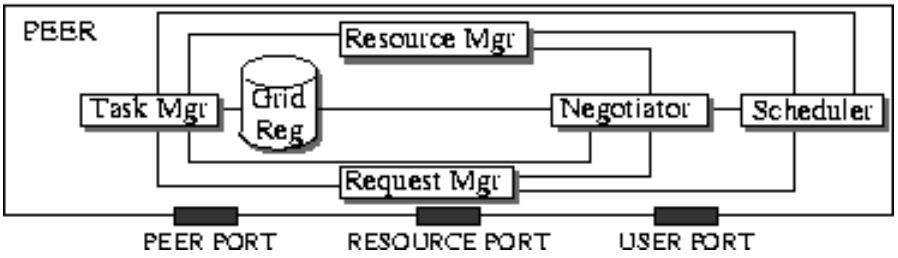


Fig. 2: Peer model

transparently for the other Peer managers. One queue stores the User Requests (i.e. local Bags of Tasks). The second queue stores the Requests made by other Peers, (i.e. external Tasks).

The supported operations on the queues are: (1) Request queueing: submitting local and external Tasks, (2) Request scheduling: selecting a local or external Task to execute it (with a FIFO policy, the oldest not currently executed Task is selected), (3) Request requeueing: reinserting a local or external Task after its execution has been cancelled, or has failed, (4) Request dequeueing: removing a local or external Task after it has been either successfully completed or cancelled, or has failed.

3.2 Resource Manager

The Resource Manager is the Peer component responsible for the management of the state of the Peer Resources. It is merely a passive component from which idle and busy Resources can be identified. The supported operations are: (1) Resource registration: as the Resources have to contact their owner Peer when they come online, and periodically afterwards indicate their continued online presence (so-called heartbeat mechanism, required by the fact that even owned Resources may be partial and intermittent), (2) Resource scheduling: random selection of idle Resources so as to schedule the execution of a Task, (3) Resource preemption: random selection of a busy (either with a local or external Task) Resource in order to cancel the execution of the currently running Task.

3.3 Task Manager

The Task Manager is the Peer component responsible for the execution of Tasks. It is an active component that uses the Request and Resource Managers to perform its operations. The supported operations are: (1) Task running, (2) Task completion, (3) Task cancellation of local Tasks to be locally run, local Tasks to be run on external Resources and external Tasks to be run on local Resources. Task execution is always dedicated, meaning that a given Resource simultaneously executes at most one Task (this does not exclude the installation of multiple instances of the Resource middleware on the same PC). Task com-

pletion operations are performed when a Resource has completed the execution of a Task: a Resource uploads the results to its owner Peer, which forwards them to the consumer Peer if the Task wasn't local.

3.4 Grid Register

The Grid Register is the Peer component responsible for the collection of management data (i.e. about its owner Peer and other Peers it has exchanged Resources with). It is a passive component composed of three databases: Grid Negotiation Profile, Grid Bartering Profile and Peer Profiles. These databases constitute a model of the Grid environment that is collected over time through interactions, without explicit information requests. These can be used by the other managers when they need to take consumption or supplying decisions involving other Peers, based on their perceived performance, reliability and negotiation behaviour.

The Grid Negotiation Profile database collects the supplying requests from other Peers so that they can be batch-processed regularly by the Negotiator. It also collects the consumption grants from other Peers so that they can be used by the Negotiator to schedule local Tasks on the Resources of external Peers. Finally, it accounts for the Negotiator activity.

The Bartering Profile database accounts for the Task Manager activity: completion, cancellation and failure of local Tasks, either locally or externally run, and external Tasks that are run locally.

The Peer Profiles database stores Profiles of all Peers with whom the owner Peer has exchanged Resources. The data stored about each Peer are Bartering data, such as Favors [4] and reliability [6] accounting, and negotiation data (supplying requests and consumption grants statistics). A different accountant can be dynamically associated with each Peer Profile. Several different resource usage accounting policies (Perfect, Time-based, Relative Power [4]) are currently implemented and more can easily be integrated.

3.5 Scheduler

The Scheduler is the Peer component responsible for the scheduling of the Tasks on local and external Peer Resources. It is an active component. The supported operations are: (1) local Tasks scheduling on local Resources, (2) generation and communication of supplying requests, (3) local Tasks scheduling on external Resources, (4) external Tasks scheduling on local Resources, (5) external Tasks preemption from local Resources.

Several scheduling policies are currently implemented (no consumption and no supplying, consumption and nonpreemptive supplying, consumption and preemptive supplying). The Peer Scheduler can be dynamically replaced if a change of policy is required.

Typically, the Scheduler will be activated on two categories of events: Job submission and Task completion/cancellation/failure. A classical sequence of

scheduling operations is: (1) local Tasks scheduling on local Resources, (2) external Tasks scheduling on local Resources, (3a) request supplying to, then reception of consumption grants from, other Peers, (3b) local Tasks scheduling on external Resources, with steps 3a and 3b repeated as long as there are both queued local Tasks and received consumption grants to enable their scheduling.

3.6 Negotiator

The Negotiator is the Peer component responsible for negotiating consumption and supplying of Resources with other Peers. It is a passive component, activated by the Scheduler. The main operations are (1) processing of received supplying requests and (2) processing of received consumption grants, which also directly involves scheduling local Tasks to Resources supplied by external Peers. Secondary operations include the sending and receiving of negotiation proposals.

Given the P2P context where there should be as few exchanged management data as possible, each Peer will rely upon models of other Peers. Therefore, the negotiation protocol does not need to be complex. Each time a Peer needs access to external Resources, it simply sends to other Peers one supplying request for s resources. The Peers that were interrogated may answer with a consumption grant which is a number c of Resources (with $0 \leq c \leq s$).

Several negotiation policies are currently implemented (no negotiation, random negotiation, favors-ranked supplying). The Peer Negotiator can be dynamically replaced if a change of policy is required.

4 Deployment

An interesting feature of the LBG architecture is the possibility to run, without any modification, scheduling and negotiation algorithms either as part of a simulator used to study their behaviour, or within an operational middleware. This is possible due to the fact that we defined Java interfaces for each component, enabling us to code two implementations with minimal effort. In other words, the Grid Fabric and Connectivity layers [7] have been virtualized. In the simulator implementation, the Peers, Resources and Users are instantiated and communicate within the same Java VM. In the middleware implementation, the components are instantiated in separate Java VM running on different computers and communicating through network calls. In both cases, the components are not aware whether their environment (other Peers, Users) is real or simulated. The important benefit of this setup is that algorithms may be accurately simulated because they are implemented only once, without any need to code a simplified simulated version.

4.1 Simulator

A discrete-event system simulator [8] has been developed. Events are inserted into an event list with respect to their timestamp. The processed events are: Job submission, Task completion and (unexpected) Task failure. Simulated Users are

activated by the simulator to submit new Jobs to the Request Manager of their Peer. Simulated Resources are activated each time a Task-related event occurs and in turn they activate their owner Peer.

After the processing of events at a given time step, scheduling operations are performed. The simulator activates each Peer in several steps. A synchronization barrier between each step ensures the negotiation process between Peers is correctly simulated.

The simulator is running in only one thread but conceptually concurrent activities, such as Peers engaged in Resource negotiation, may easily be run in a pool of preallocated threads. This allows for performance optimization without the burden of having to run every object in its own thread.

True simulation, rather than emulation, of Task execution, and the sharing of lots of common code with the middleware allow simultaneously for good simulation accuracy and performance.

Projects related to the LBG simulator include the GridSim [9] and SimGrid v2 [10] simulators, as well as a simulator developed for the development of accounting algorithms in OurGrid [4]. SimGrid is developed in C, while GridSim and the OG-related simulator are both developed in Java. The latter is the only one to specifically target P2P Grids, although the formers can handle them. GridSim and SimGrid are potentially limited by the number of simulated components because each one is run in a separate thread. Only SimGrid currently seems to share a large part of its code with a middleware toolkit, allowing to easily deploy simulated algorithms in a middleware, as the LBG simulator does.

4.2 Middleware

A middleware has also been developed from the described components, involving the implementation of communication, Task execution and Peer discovery services, all of which were easily taken care of in the simulator.

A simple communication protocol based on serialized Java objects transmitted over TCP sockets enables message passing between the entities (Peer $\leftrightarrow$ Peer, Peer $\leftrightarrow$ Resource, Peer $\leftrightarrow$ User). Each entity is running at least one daemon to process network messages. Moreover, pools of threads allow efficient handling of multiple simultaneous connections.

Arbitrary applications are easily transformed into Grid Tasks. A Grid Task is a set of classes packed into a jar file, with the requirement that the designated main class implements an interface allowing the Grid middleware to feed it input data, run it, and retrieve output data when its execution is completed.

The code of Grid Tasks is dynamically uploaded, then run on Resources. Grid Tasks are run in a separate thread which is controlled by the Resource. Input data is transferred from the Peer to the Resources and data results generated by the execution of a Grid Task are uploaded back to the Peer owner of the Resource, which forwards them to the consumer Peer if the Resource was supplied rather than locally used.

The current implementation of a Peer discovery service is basic and relies on a central directory where Peers register themselves as they come online.

5 Summary and Future Work

The Lightweight Bartering Grid architecture is a software architecture that targets P2P Grids with bartering-based resource exchange, where Peers model their environment. A Grid model presenting the target environment of Peers, Users and Resources, and their relationships was presented. A Peer model highlighted the main components of a Peer, which were then systematically detailed.

Two deployment options of the LBG architecture were discussed: a discrete-event system simulator and a middleware. An interesting feature is the sharing of much code between these two instances of the architecture. This code sharing enables fast and easy development of bartering algorithms which can be used without modification in both the simulator and the middleware.

Current ongoing work involves the development of bartering algorithms. As a future work, matchmaking involving Task requirements other than runtime-related, and better Peer discovery should be developed. It would also be interesting to gather experience by performing tests on a larger scale, out of our development lab. Finally, improved and Grid services-compliant Fabric and Connectivity layers would enable integration of this project with related projects.

Acknowledgements. We want to thank Xavier Dalem for the coding of the middleware network and task execution layers, and Valérie Leroy, Claire Kopacz and Elisabeth Spilman for their help in the preparation of this manuscript.

References

1. I. Foster & A. Iamnitchi. On Death, Taxes, and the Convergence of P2P and Grid Computing. *Proc. Workshop on P2P Systems*, Berkeley, CA, USA, 2003.
2. N. Andrade, W. Cirne & F. Brasileiro. OurGrid: An Approach to Easily Assemble Grids with Equitable Resource Sharing, *Proc. Workshop on Job Scheduling Strategies for Parallel Processing*, Seattle, USA, 2003.
3. C. Briquet & P.-A. de Marneffe. *What is the Grid ? Tentative Definitions Beyond Resource Coordination*, Tech. Report, University of Liège, Belgium 2006.
4. R. Santos, A. Andrade, F. Brasileiro, W. Cirne & N. Andrade. Accurate Autonomous Accounting in Peer-to-Peer Grids, *Proc. Workshop on Middleware for Grid Computing*, Grenoble, France, 2005.
5. B.N. Chun, Y. Fu & A. Vahdat. Bootstrapping a Distributed Computational Economy with Peer-to-Peer Bartering, *Proc. Workshop on Economics of Peer-to-Peer Systems*, Berkeley, 2003.
6. C. Briquet & P.-A. de Marneffe. Learning Reliability Models of Grid Resource Supplying, *Proc. Cracow Grid Workshop*, Cracow, Poland, 2005.
7. I. Foster, C. Kesselman & S. Tuecke. *The Anatomy of the Grid: Enabling Scalable Virtual Organizations*, Int. J. Supercomputer App., 15(3), 2001.
8. J. Banks et. al. *Discrete-Event System Simulation*, 3rd Ed., Prentice Hall, 2000.
9. R. Buyya & M. Murshed. *GridSim: A Toolkit for the Modeling and Simulation of Distributed Resource Management and Scheduling for Grid Computing*, J. CCPE, Wiley Press, USA, May 2002.
10. H. Casanova, A. Legrand & L. Marchal. Scheduling Distributed Applications: the SimGrid Simulation Framework, *Proc. CCGrid*, Tokyo, Japan, 2003.