IL NUOVO CIMENTO Vol. 32 C, N. 2 Marzo-Aprile 2009
DOI 10.1393/ncc/i2009-10401-1

Colloquia: CSFI 2008

A possible solution for dinamically managing virtual environments

M. R. Cefalà([1]), M. Mariotti([2]) and L. Servoli([1])

([1]) *INFN, Sezione di Perugia - Perugia, Italy*
([2]) *Dipartimento di Fisica, Università di Perugia - Perugia Italy*

(ricevuto il 22 Giugno 2009; pubblicato online il 16 Settembre 2009)

Summary. — In modern distributed computer systems (clusters and computing GRIDs) a new class of problems, due to the increasing heterogeneity of users' needs, have to be tackled by the administrators. One possible solution is to create on-demand virtual working environments tailored on the user's requirements. Hence the need for an architecture to manage dynamically such environments. In this work we propose a possible solution based on the use of Virtual Machines (Xen), the implementation of a Virtual Machine Manager, capable of creation, destruction and migration of the virtualized working environments. The information will be collected using a client-server mechanism, to allow the manager to deploy preconfigured Virtual Machines on the available hardware resources. When a new execution environment became active, it is automatically recognized by the Batch System Manager and is then ready to be used.

PACS 07.05.Bx – Computer systems: hardware, operating systems, computer languages, and utilities.
PACS 07.05.Kf – Data analysis: algorithms and implementation; data management.

1. – Introduction

In recent years the evergrowing need from several scientific communities for greater computing capability has led to the formation, at Departmental level of common clusters, often belonging to a computing GRID, where the hardware has been acquired by several research groups, each one with its own requirements in terms of operating systems, compilers, libraries and applications.

The drawback of this approach is the unavoidable growth of resources heterogeneity due to both the geographically dispersed nature of the organizations, and their specific needs, requirements and update schedules. Furthermore, there is the insurgence of classes of applications requiring mutually incompatible execution enviroments. This problem becomes much more difficult for those cases where resource management policies are subjected also to centralized organization as in computing grids: the adoption of operating systems and/or software packages and their updates are imposed by global

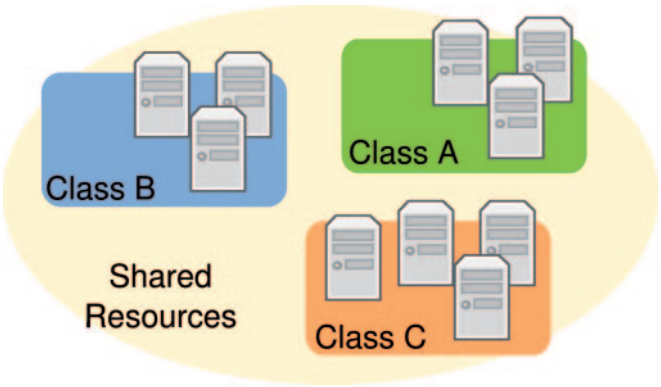

Fig. 1. – Illustration of the resource partitioning problem: three groups (A, B, C) with mutually exclusive needs have put in a common infrastructure their statically partitioned resources to ensure the availability at the expense of optimal use. Some limited management gains are still possible.

policies, but these are often not compatible with local constraints. As an example, applications developed by some of the local user groups may need long validation times before being safely ported to a new operating system or use a new library version.

The following use-case could be used to illustrate the architectural problem: three different user groups have put into a common infrastructure all their computing resources, in order to gain in maintenance manpower. Because of mutually exclusive software requests the resources have been assigned in a mutally exclusive way (fig. 1) into three sub-domains, managed by the same batch system. Since one class is not able to execute the tasks intended for another one, when there is an imbalanced workload, the resources assigned to this class are heavily loaded and can rapidly become insufficient, while the resources belonging to the other classes may be in an idle state. Beside the sub-optimal use of shared resources, there are also the management complications due to the different queues that should be set-up for the different user groups.

The desirable goal would then be to form a pool of resources that could be used by all the groups allowing them to have access to at least their share of resources and if possible, to a greater part because the time profile usage differs usually from one group to another.

2. – The Prototype's Architecture

For the reasons described previously the possibility to implement a solution, using the Virtual Machine technology, assuring to each research group its own independent working environment detached from the underlying operating system of the real machine, is becoming very attractive.

From the system administrator point of view this solution offers also the possibility to decouple the execution enviroments from the hardware resources, thus allowing to solve the problem of installing new hardware in a computing grid site where the common operating system cannot be upgraded at will to a version containing the appropriate drivers for the new harwdare.

We propose to develop a Virtual Machine Manager able to monitor the system re-

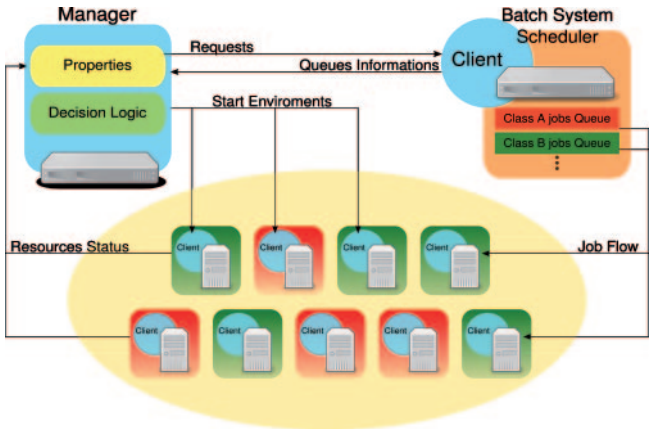

Fig. 2. – Architecture of the prototype: the Clients collect information and send it to the Manager (Server) who decides the actions to be performed, and will then send the appropriate commands back to the Clients on the various cluster resources.

sources status and to manage the Virtual Machines (create, destroy) in all the real machines of the cluster [1].

The prototype will work asynchronously in respect of the pre-existing Resource Management System of the cluster in order to do not interfere with the already existing scheduling algorithm.

In fig. 2 the Client/Server architecture of the prototype is described. The Client components will be installed on every resource, real or virtual, playing the double role of gathering information, to be passed to the Server (Manager), and of executing commands coming from the Manager. One of such resources will be the node where the Batch System/Scheduler is installed.

The information will be received by the Manager either at scheduled times or triggered by events. The information will then be analysed to check the current status of the system and the subsequent needs in terms of jobs waiting in the various queues. At the end of this phase one or more decisions would then be taken and the relevant commands (or sequence of commands) would be sent to the Clients in order to be executed, *e.g.*, the creation of a Virtual Machine tailored for group A in a real node B, where other Virtual Machines may already be running.

One last characteristic sought for the prototype is the use of OpenSource software, so we chose *Xen* [2] as the virtualization tool, *Phyton* as scripting language and *Spread Toolkit* [3] as communication system.

3. – The Communication Protocol

A crucial role in this schema will be played by the protocol that links the Manager and the Clients, because it must be robust, capable of passing the information to all Clients, with efficient and in-order guaranteed delivery capabilities.

The last point is essential because the Manager could send a sequence of commands to the same Client to have them executed in a specific order. Instead of developing ourselves the protocol we have decided to use an existing OpenSource group communication system, the *Spread Toolkit*, because of its properties that match nicely our requests. In fig. 3 a

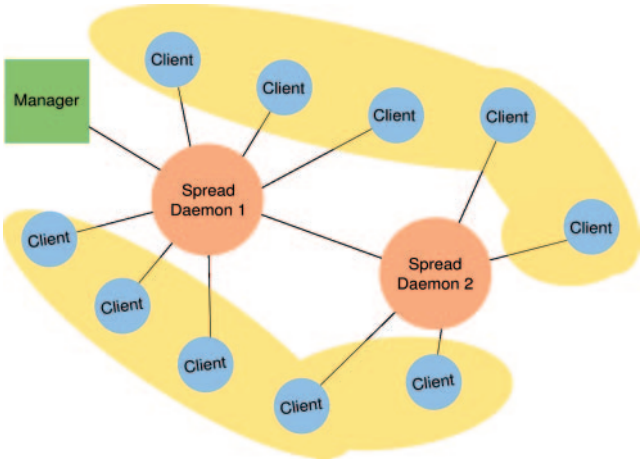

Fig. 3. – Schema of the Spread network implemented in the prototype.

schema of the Spread network to be implemented is shown; several daemons could be used if it is needed to ensure the scalability of the architecture.

Since the application is written in python, to implement the protocol we used the *SpreadModule wrapper* that allows an high level abstraction of the underlaying Spread sub-system. Two sets of messages are provided by the protocol:

– *Membership messages* generated by the Spread sub-system and received by the Manager when a Client joins or leaves a group or when it disconnects. The Manager is then able to react to status changes.

– *Regular Messages* used for properties and commands exchange. The semantic of the informative content is recognized from the *msg_type field* of the message. The Manager is able to query the Clients which will then reply sending the requested properties or the return values of the issued commands.

The content of a message (*payload*) could be anything that we want, allowing a high degree of customization that could be exploited to accommodate for future unforeseen needs.

4. – The Manager

The Manager is written in multithreaded python (fig. 4) and various threads are used to optimize the execution and avoid dead times: two threads for communication (*Listener, Sender*) and one for decision logic (*TriggerWatch*). The information extraction and the commands definition is done through property and command plugins, so there's no limitation to the type of extracted information and commands.

The Manager listens to the *Spread network*, and updates its data structures with the status of the Client groups; it also periodically sends to Clients requests for properties and aggregated data. Properties could be almost anything that we want to know; this is possible because of the flexible implementation of the protocol we discussed above.

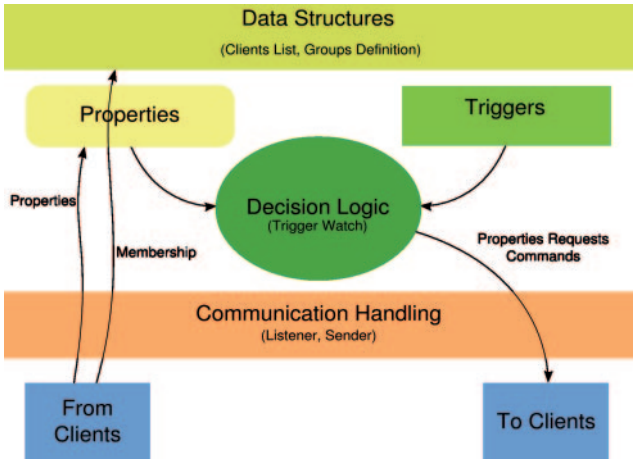

Fig. 4. – Architecturs of the Manager: the relations between the data repositories, the communication parts and the core components are shown.

An example of property definition, made of just few lines of code:

[ram_free]

command = ram

parameters = MemFree

return_type = int

ticks = 60

Ticks is the time interval between two consecutive collections of that property. An event-based mechanism, implemented through the use of *programmable Triggers*, defines the decision logic used by the Manager to choose the commands and the requests to be sent to the Clients. A Trigger definition is made of two parts:

– An event expression.

– An event-handling command.

An example:

[client_query]

expr = { len(manager.clients) != 0 }

cmd = { sender.send_get_property() }

ticks = 30

on_update_check = no

The Manager will base its decisions using the received data within a set of custom defined rules, again providing maximum flexibility to the system. The commands will then be sent to the relevant Clients to be executed on their hosts.

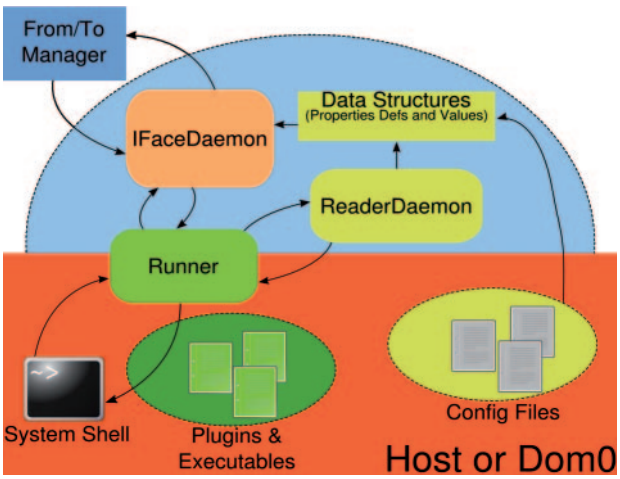

Fig. 5. – Architecture of the Client: the reletions between running environment, communication part and the two core components (*Properties Fetching* and *Command Executor*) are shown.

5. – The Client

The Client side is again written in python with a multithreaded architecture as shown in fig. 5; it is executed either in the Dom0 or DomU machines to collect all the relevant information. An interface module implements the communication protocol, while the *Runner* module allows the execution of commands issued by the manager and the asyncronous extraction of properties from the host system under the control of the *Reader* module.

6. – The Testbed

The prototype has been implemented on the University of Perugia Physics Department Computer Science Laboratory. The 35 diskless workstations (fig. 6) were equipped with Xen 3.x using a shared NFS root filesystem hosted on the boot server virtdom.

The Manager is set up to run on virtdom as well as the Spread Daemon. Each workstation is capable to run at least 50 Clients (depending on the size of RAM) for testing purposes. The interconnecting network is a standard 100 Mbit/s Ethernet.

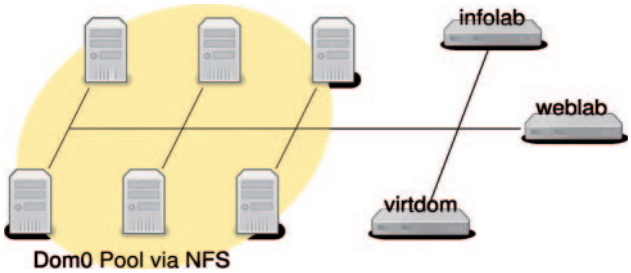

Fig. 6. – Schema of the testbed to be used for the performance tests.

Some tests have already been performed in order to evaluate:

– the basic functionality of the messaging system and of the prototype;

– the messaging protocol efficiency;

The results presently obtained shows no problems in the architecture of the prototype, all the basic functionalities having been tested. No signs of possible bottlenecks in resource usage due the communication protocol have been observed.

7. – Conclusion

A prototype to manage the creation of Virtual Environments in a middle-sized computing cluster with eterogeneous users has been designed and implemented. The driving idea of having an asynchronous system with respect to the Batch System has been explored, identifying the protocol system as one of the most important components. An implementation using the Spread Toolkit has been carried out and some tests on the scalability of the system have been performed on a dedicated testbed. The main functionalities of the prototype have been tested and no bottlenecks or unforeseen problems have been found.

More exhaustive tests concerning the scalability of the prototype will be carried out during the next months, followed by the definition of evaluation metrics in order to estimate the impact on job queue times and on the aggregated throughtput of the batch system of the prototype. A fully working prototype will be implemented in the aforementioned INFN GRID site by the end of the year 2008.

* * *

The authors would like to thank the Physics Department of Università degli Studi di Perugia for the use of the Computer Science Laboratory.

REFERENCES

[1] Cefalà M. R., *Studio di un prototipo per la gestione dinamica di ambienti virtuali*, Tesi di Laurea, Università degli Studi di Perugia (2008).
[2] `www.xen.org` Official Xen web site.
[3] `www.spread.org` Official Spread web site.