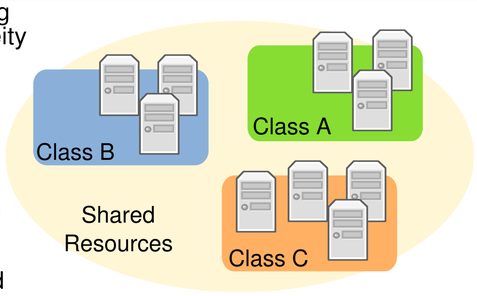
A Proposal to Dinamically Manage Virtual Enviroments in Heterogeneous Computing Facilities

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

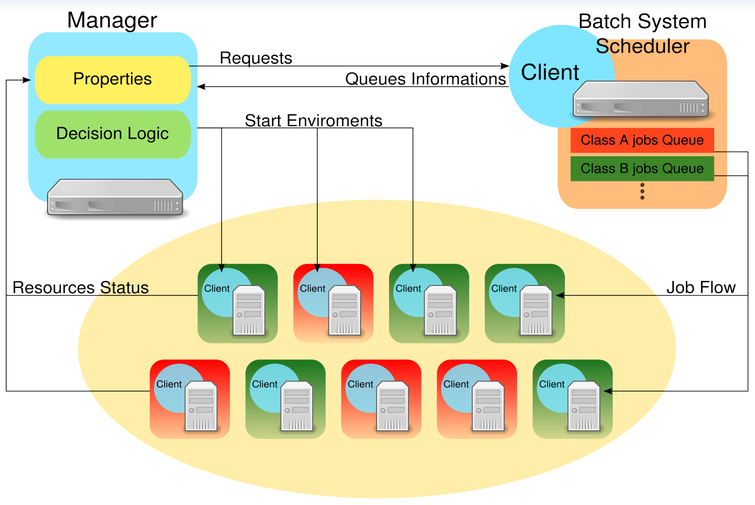
[1]INFN Perugia, Perugia, Italy; [2]Dipartimento di Fisica, Università di Perugia, Perugia, Italy;

Increasing Resources Heterogeneity and Relative Arising Problems

The implementation of modern distributed computing systems is leaning toward the ever growing practice of resource sharing among differed entities to reach high computing aggregate capabilities sharing the burden of investments, management and development. 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 policies, but there are often non trivially compatible 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.
A common solution adopted to face, at least partially, these problems is to divide the resources in classes to satisfy the various incompatible purposes. However this solution is typically the source of management complications and of the sub-optimal use of shared resources as shown in the following use-case: the resources of a distributed computing system are assigned to three classes in a mutually exclusive way.
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.



Proposed Solution



The proposed idea is to introduce a dynamic management of execution enviroments in order to have them started on-demand, according to the real requests, without modifing the resource management system.
To reach this goal it is necessary to decouple the execution enviroments from the hardware resources using virtualization techniques (*Xen*) in order to activate a new virtual machine for an incoming job request.
Some mechanisms are needed:

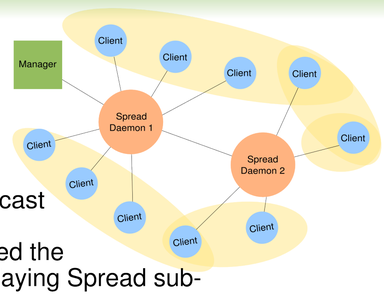
- To define execution enviroments and classes of execution enviroments.
- To monitor resource status on the system.
- To control services and execution requests.
- To decide about the activation of the execution enviroments.

A client/server application that will work side by side and asynchronously with a pre-existing resource management system has been developed, flexible enough to be used in many scenarios. The figure on the left shows the possible implementation on a batch system.
Clients will be executed on each worker node and on the batch scheduler and will be capable of extracting sensible informations from the hosting operating systems and execute commands issued by the decision logic residing on the server side called Manager.
The information extraction and the commands definition is done through ***property and command plugins***, so there's no limitation to the type of extracted informations and commands.
The Manager will query properties from Clients and will base its decisions using the received data within a set of custom defined rules. The Manager will then send the commands to the relevant Clients to be executed on their hosts.

Web References about the Xen Hypervisor: http://www.xen.org/

Implementation

Manager/Client Communication Protocol



A protocol to allow the Manager/Client interaction has been implemented using the group communication system ***Spread Toolkit***.
It provides a distributed message bus, as shown in the right figure, allowing efficient messaging with in-order guaranteed delivery.
Spread allows to define and manage groups of clients and to serve multicast and point-to-point messages.
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 (see tables) are provided by the protocol:

Membership messages are 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.

Fields	MEMBERSHIP_MESS	group	reason	extra
Type	Integer	String	Integer	String
Meaning	Message Type Id.	Group Name	Event Id.	Client Name

Regular Messages are 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 reply sending the requested properties or the return values of the issued commands.

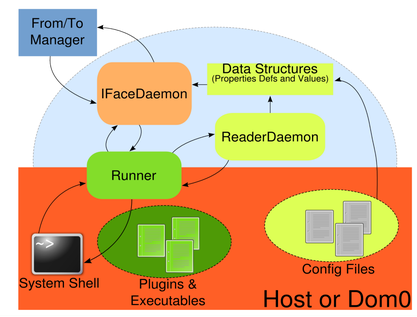
Fields	REGULAR_MESSAGE	groups	message	msg_type	sender
Type	Integer	List	String	Integer	String
Meaning	Message Type Id.	Recipients	"Payload"	Message Code	Sender

Web References about the Spread Toolkit: http://www.spread.org/

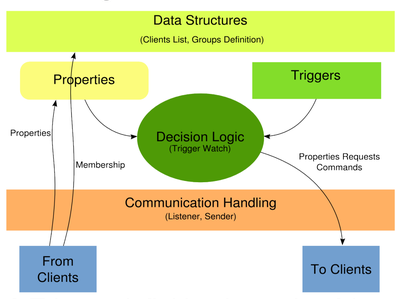
Client

The client side is written in python with a multithreaded architecture as shown in the figure on the right. An interface module implements the communication protocol.
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.
Below is shown an example of property definition:

```
[ram_free]
command = ram
parameters = MemFree
return_type = int
ticks = 60
```



Manager



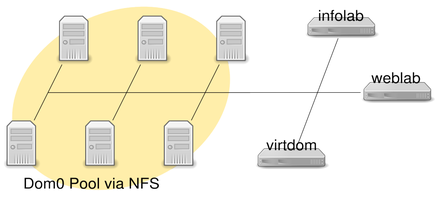
As well as the Client the Manager is written in python and consists of various threads: two for communication (Listener, Sender) and one for decision logic (TriggerWatch).
The Manager listens on the Spread network, and updates its data structures with the status of the Client groups. It also periodically sends to Clients requests for properties and aggregates data.
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 (example on the right):
- An event expression.
- An event-handling command.

Both can be implemented using python syntax. When the event expression is evaluated *true* the command gets executed.

```
[client_query]
expr = { len(manager.clients) != 0 }
cmd = { sender.send_get_property() }
ticks = 30
on_update_check = no
```

Testbed



The prototype has been implemented on the University of Perugia Physics Department Computer Science Laboratory.
The 35 diskless workstations 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 100Mbit/s Ethernet.

Performance Evaluation

Several tests have been performed in order to evaluate:
- The messaging protocol efficiency.
- The amount of Memory and CPU used by the Manager.
- The scalability of the prototype.

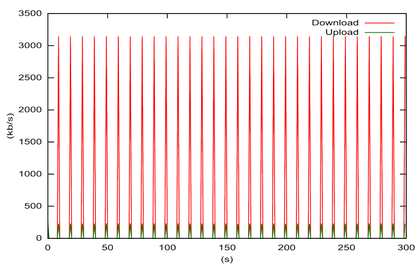
The Manager was configured to poll the clients each 10s. The Clients were set up to respond with one property of 10kb or 100kb size. It should be noticed that in a real usage scenario the messages size is much lower.
Varying the number of Clients we can see the Manager behaviour in different traffic conditions.

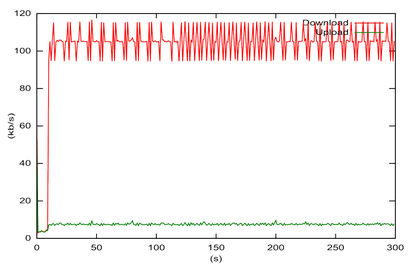
Test Results

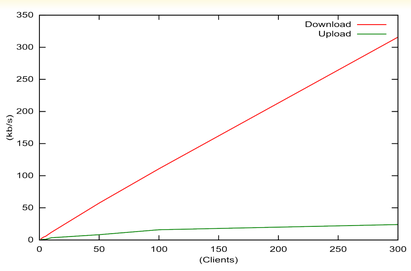
The graphs below show the usage in kilobytes per second of the network interface of the node where the Manager is running.

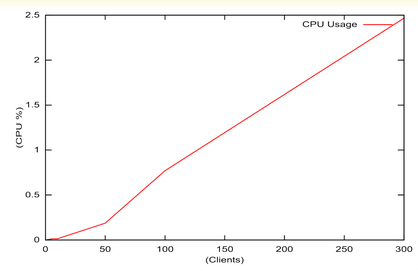
In the left graph there is no restrictions on the rate of the outgoing messages, and the spikes reach over 3000kb/s; this value is consistent with the traffic generated by 300 clients sending each a 10kb message at the same time.
In the right graph the Manager is configured to limit the rate of outgoing messages, including queries, at 10 messages per second to avoid the excessive instantaneous usage of the network by the messaging system. The results are shown for the 100 Clients case, but the same graph has been found for 300 Clients. As can be seen the download value is on average 100kb/s as expected.
The collected data show the regularity of the network traffic generated by the messaging system at an operational regime for the target use case. Is to be noticed the capability of the Manager to adapt the network usage to the eventual constraints of a congested network.

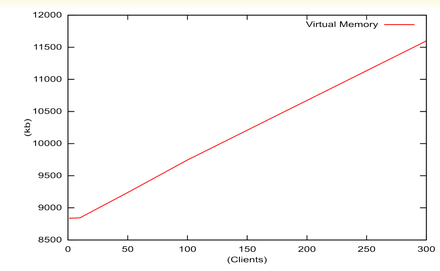
Further tests have been carried out also with 100kb message size. In this case the results are consistent with the 10kb taking into account the scale factor due to the message size. No evident bottlenecks have been found.











The three plots above show the average usage of the resources: Network, CPU Percentage and Virtual Memory. Data were obtained averaging over 5 minutes the resources usage due the Manager varying the number of Clients. The charts show that the amount of resources used by the Manager grows almost linearily in respect to the number of Clients. This means that the upward scalability possibilities of the prototype are good.

Future Works

The results obtained from the tests satisfy the usage constraints of a typical medium sized computing facility like the INFN GRID site at the University of Perugia Physics Department (~100-200 nodes). Other tests must be performed on the prototype to reach a production level reliability before it could be used in a real production enviroment.
Those further tests are aimed at the evaluation of the capabilities and performances of Virtual Enviroments Handling by the Manager according to Decision Policies.
This phase will be 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.
A fully working prototype will be implemented in the aforementioned INFN GRID site by the end of the year 2008.

Presented at IEEE NSS/MIC '08
19 - 25 October 2008 Dresden, Germany