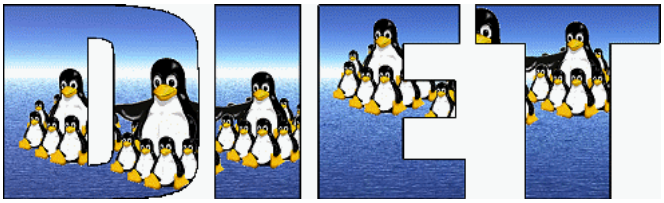

USER'S MANUAL

VERSION	2.6
DATE	April 2010
PROJECT MANAGER	Frédéric Desprez.
EDITORIAL STAFF	Yves Caniou, Eddy Caron and David Loureiro.
AUTHORS STAFF	Abdelkader Amar, Raphaël Bolze, Éric Boix, Yves Caniou, Eddy Caron, Pushpinder Kaur Chouhan, Philippe Combes, Sylvain Dahan, Holly Dail, Bruno Delfabro, Benjamin Depardon, Peter Frauenkron, Georg Hoesch, Benjamin Isnard, Mathieu Jan, Jean-Yves L'Excellent, Gaël Le Mahec, Christophe Pera, Cyrille Pontvieux, Alan Su, Cédric Tedeschi, and Antoine Vernois.

Contents

Introduction

Resource management is one of the key issues for the development of efficient Grid environments. Several approaches co-exist in today's middleware platforms. The granularity of computation (or communication) and dependencies between computations can have a great influence on the software choices.

The first approach provides the user with a uniform view of resources. This is the case of GLOBUS [?] which provides transparent MPI communications (with MPICH-G2) between distant nodes but does not manage load balancing issues between these nodes. It's the user's task to develop a code that will take into account the heterogeneity of the target architecture. Grid extensions to classical batch processing provide an alternative approach with projects like Condor-G [?] or Sun GridEngine [?]. Finally, peer-to-peer [?] or Global computing [?] can be used for fine grain and loosely coupled applications.

A second approach provides a semi-transparent access to computing servers by submitting jobs to servers offering specific computational services. This model is known as the Application Service Provider (ASP) model where providers offer, not necessarily for free, computing resources (hardware and software) to clients in the same way as Internet providers offer network resources to clients. The programming granularity of this model is rather coarse. One of the advantages of this approach is that end users do not need to be experts in parallel programming to benefit from high performance parallel programs and computers. This model is closely related to the classical Remote Procedure Call (RPC) paradigm. On a Grid platform, RPC (or GridRPC [?, ?]) offers easy access to available resources from a Web browser, a Problem Solving Environment (PSE), or a simple client program written in C, Fortran, or Java. It also provides more transparency by hiding the selection and allocation of computing resources. We favor this second approach.

In a Grid context, this approach requires the implementation of middleware to facilitate client access to remote resources. In the ASP approach, a common way for clients to ask for resources to solve their problem is to submit a request to the middleware. The middleware will find the most appropriate server that will solve the problem on behalf of the client using a specific software. Several environments, usually called Network Enabled Servers (NES), have developed such a paradigm: NetSolve [?], Ninf [?], NEOS [?], OmniRPC [?], and more recently DIET developed in the GRAAL project. A common feature of these environments is that they are built on top of five components: clients, servers, databases, monitors and schedulers. Clients solve computational requests on servers found by the NES. The NES schedules the requests on the different servers using performance information obtained by monitors and stored in a database.

DIET stands for Distributed Interactive Engineering Toolbox. It is a toolbox for easily developing Application Service Provider systems on Grid platforms, based on the Client/Agent/Server scheme. Agents are the schedulers of this toolbox. In DIET, user requests are served via RPC.

DIET follows the GridRPC API defined within the Open Grid Forum [?].

Chapter 1

Intégration continue

1.1 Plateforme de déploiement

DIET utilise plusieurs machines pour réaliser ses tests d'intégrations continues. Les machines avec l'OS Linux sont simulées avec la technologie des containeurs légers LXC. Ces dernières fonctionnent dans une machine virtuelle VirtualBox. Les machines sous windows fontionnent sous VirtualBox.

1.1.1 Cas de Diet

TODO : LXC, VM Windows ...

1.1.2 Cas de GoDiet

LXC est utilisé pour simuler plusieurs domaines (i.e plusieurs LANs).

La plateforme est composée de 3 domaines

Configuration des containeurs LXC

Le fichier de configuration doit spécifier

Exemple du fichier de configuration de la node1:

```
godiet@TestBed1:~/lxc$ cat Domain1/node1.conf
lxc.tty = 4
lxc.pts = 1024
lxc.rootfs = /home/godiet/lxc/rootfs/Lenny32/rootfs
lxc.cgroup.devices.deny = a
# /dev/null and zero
lxc.cgroup.devices.allow = c 1:3 rwm
lxc.cgroup.devices.allow = c 1:5 rwm
# consoles
lxc.cgroup.devices.allow = c 5:1 rwm
lxc.cgroup.devices.allow = c 5:0 rwm
lxc.cgroup.devices.allow = c 4:0 rwm
```

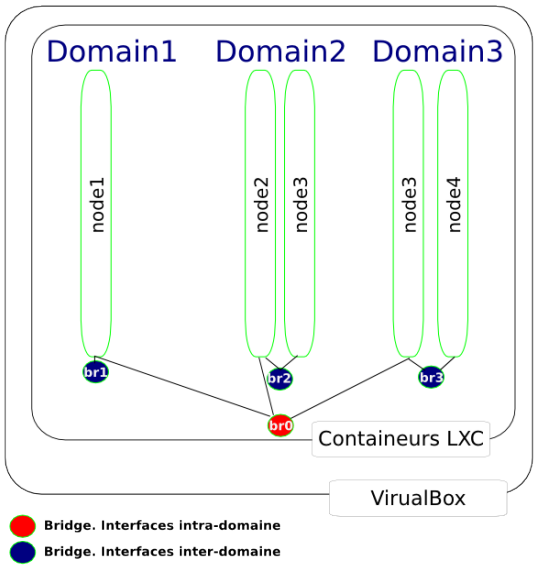

Figure 1.1: Plateforme de déploiement GoDiet.

```
lxc.cgroup.devices.allow = c 4:1 rwm
# /dev/{,u}random
lxc.cgroup.devices.allow = c 1:9 rwm
lxc.cgroup.devices.allow = c 1:8 rwm
lxc.cgroup.devices.allow = c 136:* rwm
lxc.cgroup.devices.allow = c 5:2 rwm
# rtc
lxc.cgroup.devices.allow = c 254:0 rwm

# mounts point
lxc.mount.entry=proc /home/godiet/lxc/rootfs/Lenny32/rootfs/proc proc nodev,noexec,nosuid 0 0
lxc.mount.entry=devpts /home/godiet/lxc/rootfs/Lenny32/rootfs/dev/pts devpts defaults 0 0
lxc.mount.entry=sysfs /home/godiet/lxc/rootfs/Lenny32/rootfs/sys sysfs defaults  0 0

lxc.utsname = node1

lxc.network.type = veth
```

```
lxc.network.flags = up
lxc.network.link = br1
lxc.network.name = eth0
lxc.network.ipv4 = 192.168.1.11/24

#Internal domain net interface
lxc.network.type = veth
lxc.network.flags = up
lxc.network.link = br0
lxc.network.name = eth1
lxc.network.ipv4 = 192.169.1.1/24
```

Pour enregistrer une machine auprès de LXC:

```
lxc-create -n node1 node1.conf
```

Pour enregistrer une machine auprès de LXC: Pour démarrer un containeur enregistré:

```
lxc-start -n node1
```

Rajouter l'option -d pour lancer le containeur en tâche de fond.
Pour arrêter un containeur:

```
lxc-stop -n node1
```

Pour lister les containeurs ainsi que leur état:

```
godiet@TestBed1:~/$ lxc-info -n node1
'node1' is RUNNING
```

Configuration des bridges

```
#BR0 interdomain
tunctl -t tap0
brctl addbr br0
brctl addif br0 tap0
ifconfig br0 192.169.1.254

#BR1 domain1
tunctl -t tap1
brctl addbr br1
brctl addif br1 tap1
ifconfig br1 192.168.1.1

#BR2 domain2
tunctl -t tap2
brctl addbr br2
brctl addif br2 tap2
```

```
ifconfig br2 192.168.2.1

#BR3 domain3
tunctl -t tap3
brctl addbr br3
brctl addif br3 tap3
ifconfig br3 192.168.3.1
echo "Perhaps need promisc up or tapX up"
```

Configuration du domain hôte

Pour donner l'accès à l'extérieur au domaine invité:

```
Sur le domaine 0:
sysctl -w net.ipv4.ip_forward=1

iptables -A FORWARD -i br1 -j ACCEPT
iptables -A FORWARD -o br1 -j ACCEPT
iptables -t nat -A POSTROUTING -o eth0 -j MASQUERADE
```

Ne pas oublier de rajouter une route sur le domaine invité ainsi que le resolv.conf pour les DNS