Thin-OSCAR : Design and future implementation

Benoît des Ligneris [a] [b][*], Michel Barrette[a], Francis Giraldeau[a][b], Michel Dagenais [b]

[a]Centre de Calcul Scientifique,
Université de Sherbrooke, Québec, Canada

[b]Département de génie informatique,
École Polytechnique, Montréal, Québec, Canada

A description of the actual status of the thin-OSCAR project will be made. Future development of the project will be detailed and desired functionality will be exposed. We will then determine and analyze the interactions that are needed with the different components of OSCAR.

Une description du statut actuel du projet thin-OSCAR sera faite. Enfin, les grands axes de développement du projet seront discutés ainsi que les fonctionalités additionnelles souhaitées. Une analyse des interactions requises avec les différentes composantes d'OSCAR sera proposée.

Introduction

While OSCAR [1] has been conceived for clusters with disk(s) since its very first version, diskless (and systemless) support was a feature that lot of people expected. The Center for Scientific Computing [2] (CCS) has built several clusters without disk [3]. We tested OSCAR and were easily convinced of it's quality and professionalism especially compared to our own home-made scripts. We decided to use OSCAR for our diskless cluster and then transfer our diskless expertise to the OSCAR project. The thin-OSCAR [4] work-group was created to specifically analyze and solve problems that will arise while adding this diskless clusters support.

We will first define essential notions for a diskless cluster. Then, the actual implementation of thin-OSCAR will be exposed. We will expose a « road-map » for the development of thin-OSCAR. Interaction with OSCAR will be detailed so that features can be discussed, prioritized and eventually added to the OSCAR framework.

1 General definitions

This section presents some useful notions from a node perspective and several existing technical solutions for diskless support.

1.1 Node Definition

We planned to support several types of nodes, from nodes without disk to nodes with at least a disk dedicated to the cluster. In this section, we will define the node thickness levels.

1.1.1 Diskless nodes

Diskless nodes are well named : they have no disk. The main consequence is that the node has no functionality to boot without the presence of a network mechanism to initiate the boot process and then provide necessary permanent storage (disk) over the network. This kind of node is generally found in dedicated diskless clusters and in network terminals [5]. However, nodes with disks can be considered diskless when, for security or practical reason, the disk cannot be used. This can be the case in a GRID environment where cluster nodes are workstation during the day for instance. It allows nodes that are not used in a cluster to be very rapidly integrated into a cluster without any alteration of the "main" operating system stored on their disk.

Diskless clusters maybe more efficient, affordable and reliable than clusters with disks. Limitations on the type of computation that can be made on this kind of nodes exist : intensive I/O applications could be executed on such nodes but obviously, will not scale well and slow down each calculation to the point it is extremely inefficient. It will even crash the master node, if the protocol used is not robust enough.

In order to limit or even avoid I/O intensive tasks on this kind of node, a special property must be defined so that the queuing system can manage efficiently those nodes (property proposed name : `diskless`).

1.1.2 Systemless nodes

Those nodes have a file (swap file, temporary storage), a partition (swap, temporary storage) or a complete disk dedicated to the cluster ; they don't contain a disk bootloader, and boot from the network. There is large variety of nodes and no more formal definition is necessary.

This type of node is complicated to manage and special precautions must be taken : some useful data may already be stored on the disk (even alien operating systems) and because they are not completely diskless, the disk has to be accessed with care, especially when installing the node (initialization of the swap file, copy/rsync of directories, formating of partition). If those nodes have swap space and temporary storage, then they have no particular property for the queuing system, otherwise they will have the same property as the diskless nodes (`diskless`).

[*]Corresponding author : benoit@des.ligneris.net

1.1.3 Diskfull nodes

Those nodes have dedicated disks for the cluster. This is the only type of node currently managed by OSCAR. Management of nodes can be done entirely within the OSCAR framework : installation from scratch of the nodes, including hard disk formating, or update of the image (via the C3 [6] command **cpushimage** command).

1.2 Diskless and systemless clustering solutions

Several solutions exists to boot a node without disk. We will not discuss here the different booting mechanisms (PXE, floppy, ...) but the different models that exist so that nodes can be functional.

The main idea behind diskless solutions is that disks are useless for numerous types of calculation. Because network is a prerequisite in order to have a cluster, you can get rid of disk by using a network boot process. Moreover, disks are a point of failure, and because you must store nodes data onto permanent storage, like SAN, disks on nodes brings only trouble.

1.2.1 Root-NFS model

To the best of our knowledge, this is the oldest solution. The NFS protocol is very robust and can be used, if the kernel supports it, to initialize and run a system directly from the network. It does not solve the first step (i.e. how to transfer the initial kernel !) but, along the years, this method proved its viability and reliability [3].

Its main drawback is a scaling problem that is common to all clusters that share files with the NFS protocol (`/home` is generally exported and used like a distributed resource among the nodes). As a consequence, if computations make intensive I/O usage, network will be exclusively used by the NFS protocol, and then, the cluster will be paralyzed and finally crash the system (NFS server), depending on the quality of the NFS implementation.

A common solution to this problem is to use a dedicated network exclusively for the transport of information for permanent (NFS) storage. However, it does not solve the inherent NFS problem : server is central and NFS (network) load can't be distributed. While this problem wasn't very important when building small clusters, nowadays, it is a very important problem as clusters are commonly built with more than 1000 nodes.

In that respect, diskless clusters have the same problem but it occurs on a smaller scale because NFS is more heavily used. The complete root file system of each nodes resides on the NFS server. Only /opt and /usr are common (and read only). As such, diskless nodes are not good candidates for large root-NFS based clusters.

1.2.2 Ramdisk model

With this approach, a minimal ramdisk containing a completely functional Linux system is uploaded to the client. It's root device is in RAM. Once this is done, it can mount NFS partitions or any distributed file-system partition in order to access programs and users data. Once this is done, some RAM is used by the root-RAM-disk (see [10] for a more complete description of the process). The typical footprint of the minimal ram disk is $20MB$ and it involves either recompiling your kernel or making some root-raid-ramdisk.

The main interest of this approach is that the transfer of the initial ramdisk can be multi-casted so that booting a cluster can be very fast (this is not possible with NFS). Another advantage is that the connection with the file server can be lost and the node will still be up and running correctly (as the `/var` partition is included in the ramdisk).

1.2.3 Network Block Device (NBD) model

This is a relatively new distributed file system, compared to NFS (first version around 1997) [11, 12] that seems very interesting for clusters. NBD devices cannot be shared unless readonly but they can be duplicated and, as a consequence, load balancing and even redundancy can occur so that the network will be used much more efficiently. It allows much better scaling : you can add a NBD device on another file server so that your network does not saturate as you add nodes to your cluster. The team isn't aware of any clustering solution that use this file system.However, replacing NFS is definitely needed so that a better scalability can be achieved.

1.2.4 Single System Image Model

This is a whole new class of clustering. It doesn't need to be diskless but this model supports diskless clusters naturally. The idea behind those implementations (e.g. [13–16]) is to simplify both the administration and usage of clusters. It is the same idea behind SMP computers : the whole cluster appears as a single resource with lots of CPU and RAM. Efficient algorithms allow either queuing or load balancing. Several clustering distributions exist that are built with diskless nodes in mind and function very well with disks (for swap, temporary storage or distributed file-system).

2 Actual Implementation

The actual implementation of thin-OSCAR is very simple and is more a proof of concept than anything else. It is based on Ram Disk model. The script is a non-interactive Perl script **oscar2thin.pl** which operates in a serial way, transforming step after step a regular systemimager image into a RAM disk for diskless operation. Then, files are moved to their relevant location (`/tftpboot`) and some configuration files are modified on the server so that NFS export and PXE operations are adapted for the diskless cluster. We will examine each of these steps.

While the implementation details of those steps are expected to change, general ideas are expected to stay as development progresses, adding more user-friendliness and options in the process. Therefore, it seems interesting to detail those functions.

2.1 Image creation

We provide some reduced-size image into OSCAR in the `oscarsamples/Mandrake-8.2-noX-i386.rpmlist` and `oscarsamples/Mandrake-9.0-noX-i386.rpmlist` which are rpm lists with a minimum number of RPMs : no Xfree86 and no compiler on the node. This image list can certainly be reduced but those files are a good starting point.

2.1.1 Blank image creation

Loopback device support has to be available on the system (master node) where this script is executed. ”loop” is the module that enable the loopback device support. The script creates a traditional `ext2` file-system as a loopback device. The size of this partition has to be defined otherwise it is $80Mb$ which should be sufficient in most cases. Note that this maximum size will not be used in RAM (Ramdisk uses only their exact size) nor be transfered via the network (the image is compressed in the process). This file-system is then mounted in a temporary location called image directory.

2.1.2 Image copy and customization

The relevant directories are copied from their systemimager image location to the image directory : `/bin`, `/boot`, `/dev`, `/etc`, `/home`, `/lib`, `/mnt`, `/proc`, `/root`, `/sbin`, `/var` (some of the copied directories are empty). The directories `/usr` and `/opt` are created : they will be used as mounting point for the NFS exported file-system.

The `/etc/fstab` file is then generated in the image directory and contains the `/dev/ram0` device as its root mount point, the `/home` is mounted via NFS (standard OSCAR) and the `/opt` and `/usr` directories are NFS mount points from the systemimager image directory on the server.

Networking capabilities are then generated (mainly `/etc/sysconfig/network-scripts/ifcfg-eth0` which is configured via DHCP).

2.1.3 Image cleaning

This step is interesting because it allows us to reduce considerably the image size. At this step, two optimizations are made. First, we delete the modules directory (at least $10Mb$ in recent Linux distributions) and the RPM database (no further RPM operation on the node).

2.1.4 Server “configuration”

The `/etc/hosts` file is copied in the image directory, `/var/spool/pbs/server_name` and `/var/spool/pbs/mom_priv/config` generated in the image directory so that PBS will be functional on the nodes.

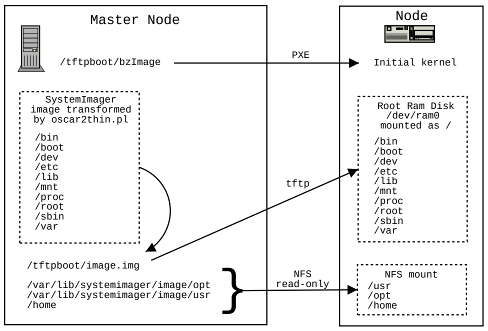

Figure 1. Master node and node scheme

2.1.5 Image compression, installation, PXE and NFS configuration

The image is then unmounted, compressed (because default kernel can uncompress ramdisks on the fly) and copied in the `/tftpboot` directory so that it can be transfered by TFTP. The default configuration file for PXE is created (`/tftpboot/pxelinux.cfg/default`) so that PXE loads the kernel `/tftpoot/bzImage` and the ramdisk we just created.

The `/etc/exports` is adjusted so that all the systemimager image is exported (read only) to the cluster net with a given subnet mask. `/home` is exported read-write to the same network.

3 Development road-map

The main goal of the thin-OSCAR work-group is to add diskless and systemless support to the OSCAR clustering framework. As a consequence, support for all the models (root-NFS based, NBD based and single system image based) of diskless and systemless nodes is expected. This will lead to profound modification or at least strong interactions with some core components of OSCAR that we will try to anticipate in the next section. This road-map is a more detailed version of the `oscar-package/ROADMAP` file.

3.1 Kernel discussion

For the moment, you need to compile your own kernel (sample configuration file provided in the `sample/ramdisk.cfg` file). We plan to distribute some kernels with the thin-OSCAR package. It is the simplest solution as we can not provide support for any installation and configuration. General NIC will be supported (eepro100, rtl8139, ...) and, of course, configuration file provided so that exotic hardware can easily be supported (with a kernel recompilation).

Modular kernel is another possibility that is already used in LTSP [5]. The potential number of modules needed by a diskless node is small : network adapters and (distributed) filesystems are the two main classes of necessary drivers. Moreover, we will focus our development effort on nodes with i586 or higher and PCI bus. Only the necessary modules will be included in the image in order to keep the image small. Other modules can be accessed via a distributed file system and only those from the two mentioned classes (NIC and filesystem) are indispensable for the successfull network boot of the node.

Because we will distribute our kernel, we will be able to add support for very different clustering solutions :

– root-NFS, ramdisk or NBD based diskless/systemless nodes
– distributed file systems (lustre, XFS, ...)
– automatic load balancing (openMOSIX)
– trace capability (Linux Trace Toolkit)
– real-time Linux (RTAI)

3.2 version 0.0-0.5 : Proof of concept

This is the actual version of thin-OSCAR.

3.2.1 Features

– All nodes DHCP
– Homogeneous cluster only : all nodes share the same image
– Monolithic kernel only
– Sample kernel configuration file provided

3.2.2 To-do

– Add a button in the OSCAR wizard for making thin-OSCAR nodes
– Pre-compiled kernel (1 standard, 1 SMP) with the most common NIC drivers built-in.
– Distro independence : most of what the **oscar2thin.pl** script is doing right now is very distro-dependent. We want to be distro independent and will certainly write a distro abstraction layer in the process.

3.3 0.5-1.0 : OSCAR integration

– Well tested and functional version for complete diskless cluster
– Support for mixed cluster : **C3** interaction !
– Storage of node information in order to provide heterogeneous cluster support (each node can use a different kernel or a different image). SIS integration ?
– Command Line Interface :
  – Creation of a complete diskless cluster with a simple file having the following information : MAC,IP,node name,KERNEL,SISIMAGE
  – Automatic collection of all MAC address : completely automatic install

3.4 1.0-1.5 : Heterogeneous cluster support

– Meta-definition of diskless model (name of model, kernel to use, SIS image to use, DISK usage [none, /tmp, /var, /home, ...], modules to keep [drivers/net], configuration file generator in order to fill configuration files [/etc/X11/XF86config-4, ...].

The aim of this step is to support heterogeneous clusters : disk/no disk, swap file, swap partition, /tmp storage, distributed /home, ...

3.5 2.0 : long term objective / wish-list

– provide last generation kernel for diskless nodes with precompiled utilities (openMosix, LTT, lustre, coda, ...) so that tests can easily be made and other types of clusters can be supported.

4 OSCAR interaction

In this section we will examine what kinds of additional features are needed by OSCAR to support thin-OSCAR. When possible, the actual component of OSCAR that is involved will be identified and some solutions proposed. Some items are more important than others and this will help us to prioritize OSCAR and thin-OSCAR development.

4.1 Storage of additional thin-OSCAR information : OSCAR Database

As explained in the first section of this article, there is several kind of nodes and the ultimate goal of thin-OSCAR is to support all the nodes. Nodes properties have to be carefully determined. As it can be used by several OSCAR packages, this data should be organized in a relational database of well defined structure and format. While OSCAR Database provides a database abstraction and a permanent storage area for OSCAR packages, all the common information has to be organized and a GUI has to be created to enter this data (MAC address, model, ...).

In order to support diskless and systemless clusters, the installation process of an image has to be refined. Indeed, systematic format of hard disk prior to installation has to be avoided.

4.2 Image diffusion and build : systemimager ?

This is one of the steps made by the **oscar2thin.pl** script. As we use mainly SIS information and certainly redo (very crudely) what SIS is already doing (namely network and file-system configuration), maybe a kind of integration should occur in order to ease the development of both SIS and thin-OSCAR.

4.3 Cluster Management : C3

C3 is a core component of OSCAR and it works very well with nodes with disk. In order to support diskless (and systemless) clusters, the **C3** tools have to be aware of diskless nodes. Several commands have to be modified :

– **cpush** Instead of remote copying the file, the file has to be copied in the relevant SIS image directory
– **cpushimage** A new initial RAM disk has to be regenerated and nodes using this image has to be rebooted
– **crm** has to remove files in the SIS image directory

Morevover, warning should be send to the user because he can't modify a single node : those actions will affect the group of nodes that use the altered image.

4.4 Different kinds of nodes

This is a crucial point. Currently, nodes can be "personalized" only with the selection of different rpmlist for them. However, in practice, there is one (or two !) file available for each distribution. We need a more complete way of managing different kinds of nodes.

A mechanism to store, distribute and manage node "profiles" has to be created within OSCAR. The potential of this point is very important for the openness of OSCAR. Here are some arguments in favor of OSCAR node modelization :

– Separation of the master node in several nodes in order to achieve a better scaling : dedicated NFS server, dedicated PBS server, dedicated monitoring server, ...
– Cluster hierarchy : OSCAR currently supports two levels of nodes (one master and several nodes). For large clusters, this is not suitable and we should be able to add at least one level. However, implementing one additional level is not much more difficult than adding a general node abstraction layer.

Conclusion

Useful definitions of nodes were exposed : diskless, systemless and diskfull. Then some techniques for diskless and systemless clustering were briefly reviewed. Finally, the goals, actual implementation and future development of thin-OSCAR were exposed.

The last part discussed interactions with OSCAR, OSCAR modifications and finally new OSCAR features that are needed in order to fulfill the thin-OSCAR goals. Even if this is a written article, most of what is exposed here is open for discussion and not carved in stone !

Acknowledgments

We would like to thank Michel Barrette, Mehdi Bozzo-Rey and Sean Dague for valuable discussion. All this work was possible thanks to the existence of the Center for Scientific Computing of Sherbrooke University [2] and its enthusiastic acceptance of the Open Source model.

References

1. Open Source Cluster Application Resource (OSCAR) `http://oscar.sf.net/`
2. Centre de Calcul Scientifique, Université de Sherbrooke `http://ccs.usherbrooke.ca/`
3. Development, installation and maintenance of Elix-II, a 180 nodes diskless cluster running thin-OSCAR, M. Barrette, M. Bozzo-Rey, C. Gauthier, F. Giraldeau, B. des Ligneris, J.-P. Turcotte, P. Vachon, A. Veilleux, submitted to HPCS2003.
4. thin-OSCAR work-group. Support for diskless and systemless cluster for OSCAR `http://thin-oscar.ccs.usherbrooke.ca/`
5. Linux Terminal Server Project `http://www.ltsp.org/`
6. Cluster Command & Control (C3) power tools, `http://www.csm.ornl.gov/torc/C3`
7. NFS-Root mini-HOWTO `http://www.tldp.org/HOWTO/mini/NFS-Root.html`
8. TFTP standard (RFC 1350) `ftp://ftp.rfc-editor.org/in-notes/rfc1350.txt`
9. System Installation Suite website `http://www.sisuite.org/`
10. Root Raid in Ram How-To, Mehdi Bozzo-Rey, Michel Barrette, Benoît des Ligneris submitted to HPCS2003 (2003).
11. Network Block Device project page `http://nbd.sourceforge.net/`
12. The Network Block Device, *Linux Journal*, 73, may 2001 `http://www.linuxjournal.com/article.php?sid=3778`
13. Scyld Beowulf Scalable Computing `http://www.scyld.com/`
14. Beowulf Distributed Process Space `http://bproc.sourceforge.net/`
15. Single System Image Clusters (SSI) for Linux `http://openssi.org`
16. openMOSIX `http://www.openmosix.org/`