Utilisation d'un cluster de calcul openMosix et déploiement à l'aide de LTSP

Maurice Libes
Centre d'Océanologie de Marseille - UMS 2196 CNRS & CISCAM de l'Université de la Méditerranée
Campus de luminy - case 901 13288 Marseille cedex9
maurice.libes@com.univ-mrs.fr

Résumé

L'article propose de présenter l'utilisation de deux distributions complémentaires dans deux domaines très différents :

- *openMosix est un système dit de « clustering » permettant de créer une grappe de PC (cluster) dédiée au calcul, qui se comporte quasiment comme une machine multiprocesseurs SMP. openMosix permet d'accroître la puissance de calcul en mettant en commun les processeurs de plusieurs machines reliées en réseau. Les algorithmes de openMosix travaillent au niveau du noyau Linux et permettent de faire collaborer les machines ensemble en échangeant des processus et en répartissant la charge dans le cluster.*
- *LTSP est un système de « serveur de terminaux » qui fournit disque et système à des PC sans disque, permettant ainsi le déploiement efficace de larges ensembles de PC avec une administration extrêmement facilitée.*

Chacun de ces deux systèmes possède son propre domaine de prédilection et peut bien sur être utilisé séparément. Cependant, lorsqu'on combine l'utilisation de openMosix et de LTSP, avec quelques services Unix complémentaires (comme tftp, dhcp, nfs, xdm), on crée une architecture matérielle et logicielle permettant non seulement de créer un anneau de calcul mais en plus de le déployer et le faire varier de manière souple et efficace. Avec une simple disquette Etherboot ou d'un boot automatique par PXE, on peut rajouter à volonté des PC dans l'anneau de calcul et augmenter ainsi la capacité de calcul.

L'architecture openMosix+LTSP procure une grande souplesse et facilité d'administration ainsi qu'une grande efficacité. Elle permet d'obtenir une fermette de PC facilement extensible pour un coût budgétaire et une administration très restreints, pouvant répondre efficacement à des besoins de calcul dans bon nombre de laboratoires.

Mots clefs

openMosix, LTSP, cluster de calcul, grappe de PC, migration de processus, équilibrage de charge, serveurs de terminaux, clients légers, déploiement de systèmes, etherboot, pxe, administration centralisée, OpenSource.

1 Introduction

1.1 Le contexte

En quelques mois, une dizaine de chercheurs arrivent progressivement dans votre laboratoire pour travailler sur la modélisation numérique de processus biologiques et physiques. Chacun d'eux possède un PC et accueille des étudiants qui vont modifier plusieurs codes de calcul, les développer et faire des simulations qui vont produire des gigaoctets de résultats. Les PC sont très hétérogènes : aucun n'a été acheté au même moment ni chez le même fournisseur. Certains ont les crédits pour s'acheter la meilleure machine du moment (disons à 4000Euros pour l'exemple), alors que d'autres (étudiants) ont de vieux Pentium II 64Mo de RAM, et 16Go de disque. Bien entendu vous devez les aider dans leurs projets scientifiques en administrant toutes leurs machines. Les demandes fusent : installation de 2 systèmes en dual-boot (Windows et Linux), compilateur Fortran 90, Fortran77 ou « C » (j'ai échappé à ADA), multiples échanges de fichiers entre les PC, création de plusieurs points de montages croisés NFS... sauvegarde des projets... installation d'outils de développement complémentaires : Matlab sur trois PC , PVwave sur 4 autres... Visionnaire, vous savez que les 10 meilleurs PC du moment seront relégués aux oubliettes dans quelques mois, et qu'il faudra 40000Euros supplémentaires 3 ans plus tard pour les renouveler afin de combler l'évolution des besoins (et parfois satisfaire les « egos » de chacun).

Cette situation difficile à gérer sur le plan technique et financier, est souvent la conséquence de plusieurs effets conjugués :

- montée en charge progressive (les équipes se sont constituées au fil du temps)
- absence totale de schéma directeur en matière de calcul scientifique

- absence de prise en compte globale des coûts (humains et informatiques) dans le laboratoire
- méconnaissance des outils informatiques et des produits libres openSource chez nombre d'utilisateurs
- absence de prise en compte des nécessités fondamentales de l'administration des systèmes informatiques.

On comprend aisément que la gestion de cette configuration informatique décentralisée ne peut être que très médiocre. Elle demande à l'administrateur de décupler ses interventions et d'obtenir un rapport « efficacité d'intervention/énergie dépensée » insatisfaisant pour les utilisateurs comme pour les administrateurs. Dit autrement, « les coûts humains et budgétaires sont maximaux ! »

Confrontés à ces problèmes de multiples interventions d'administration erratiques, la plupart des administrateurs systèmes mettent en avant (Milhaud [1]), qu'il est nécessaire de centraliser les équipements et l'administration des systèmes. Certains se tournent vers l'utilisation de « clients légers » (Auroux et al. [2]) afin d'améliorer l'efficacité des interventions techniques et réduire les coûts budgétaires. C'est dans cette optique que nous avons cherché une solution efficace et si possible économique qui puisse répondre de façon commune aux besoins de calcul des chercheurs, tout en réduisant les efforts d'administration. Pour mettre en place une configuration de calcul centralisée, nous nous sommes tournés vers des systèmes de gestion de « *cluster de calcul* » (encore appelé «grappe de PC» ou « fermette de PC »). Nous verrons dans cet article que openMosix est une réponse techniquement efficace et financièrement économique, à ces problèmes de besoins de calcul.

Une fois openMosix installé et testé sur un PC, nous nous sommes intéressés aux possibilités de déploiement du cluster par des logiciels de clonage classique, mais surtout avec LTSP qui est un système de « serveur de terminaux ». A partir d'un poste serveur, LTSP fournit automatiquement à des PC clients les disques et le système dont ils ont besoin. On verra que cela est suffisant pour faire entrer un PC dans l'anneau de calcul openMosix. Il n'y a donc aucune charge d'administration subséquente sur les postes clients. L'effort d'administration est porté sur la seule machine serveur.

En combinant l'utilisation de openMosix et de LTSP, nous montrons comment on déploie facilement la taille du cluster. Pour terminer, nous discutons des avantages et des inconvénients de openMosix et LTSP.

2 openMosix un cluster HPC

Le terme « cluster » recoupe plusieurs définitions ou fonctionnalités selon la ressource qui est partagée par les ordinateurs. On peut distinguer différentes classes de clusters :

- les clusters de calcul scientifique (HPC High Performance Computing). Il s'agit d'un cluster où les noeuds cumulent leur puissance de calcul pour rivaliser avec les performances de super-calculateur. Le but d'un cluster de calcul (HPC) est de faire coopérer les ordinateurs entre eux au sein d'un réseau en partageant les processeurs et la mémoire. On améliore ainsi les temps de calcul par rapport à des programmes qui exploiteraient localement le processeur et la mémoire d'une seule et même machine.
- les clusters « haute disponibilité » « HA High Availibility » on cherche ici à obtenir de la fiabilité par une redondance de machines. Chacune pouvant prendre le relais de l'autre en cas de panne. Dans ce cas le fonctionnement du cluster et l'assurance contre les pertes de données sont garantis au maximum.
- Les clusters de stockage : on cherche ici à grouper les espaces de disques pour stocker d'énormes fichiers.
- Les cluster de répartition de charge (architecture de serveur Web par exemple)

openMosix est un cluster de calcul de type HPC, un peu comme ceux de type Beowulf *(http://www.beowulf.org)*. Cependant les clusters Beowulf fonctionnent avec des applications qui, pour tirer bénéfice du cluster, nécessitent d'être liées à des librairies de parallélisation particulières comme PVM [3] ou MPI. Ils demandent donc aux utilisateurs de savoir programmer leurs applications avec des appels à ces librairies, ce qui limite donc leur utilisation à une population déterminée. On trouvera dans Bar[4] une étude comparative entre openMosix et Beowulf. Le système openMosix est différent par le fait qu'il réalise les opérations de répartition de charge et de synchronisation des machines de façon transparente et automatique vis à vis de l'utilisateur et des applications. L'exploitation d'un cluster openMosix ne nécessite pas de modification des applications (Le Cannelier [5]) et aucune connaissance particulière en programmation PVM ou MPI n'est requise pour bénéficier de la répartition de charge au sein du cluster (Robbins [6]).

2.1 Principes de base de openMosix : migration de processus et répartition de charge

openMosix est un système de gestion de cluster. Il permet à un ensemble de « n » PC hétérogènes reliés en réseau, de coopérer et de travailler ensemble comme si c'était une seule et même machine multiprocesseurs (SMP) à mémoire partagée.

Par exemple, du point de vue de l'utilisateur, openMosix donne l'illusion d'avoir une machine octo-processeur en lieu et place de 4 machines bi-processeur.

Dans un cluster openMosix, le plus petit élément gérable est le «processus» (image du programme en cours d'exécution en mémoire). La coopération entre les noeuds du cluster se fait au niveau du « processus ». Le principe de base de OpenMosix est de faire migrer un processus de façon dynamique et préemptive vers le noeud du cluster qui présente les meilleures caractéristiques et performances à un temps « t ». L'expression consacrée par les concepteurs de Mosix est « *Fork and Forget* »). La stratégie sous-jacente est de migrer un processus après avoir calculé un « coût » prenant en compte les caractéristiques de chaque noeud : CPU, mémoire, bande passante) en une valeur unique. Les processus sont alors affectés à la machine présentant le meilleur coût.

La principale caractéristique de openMosix est la recherche automatique d'une performance optimale pour l'exécution des processus. En effet lorsque des processus sont créés, openMosix calcule et détermine automatiquement quel est le noeud le moins chargé en terme d'utilisation du processeur et d'occupation de la RAM, et fait migrer une partie du processus en cours d'exécution vers la meilleure machine : c'est à dire celle qui offre le processeur le plus rapide, le moins occupé et la mémoire la moins saturée. On trouvera dans Barak, La'adan [6] et Bar [7] les caractéristiques principales de Mosix :

- un « *mécanisme de migration des processus* » préemptif : La sélection des processus à migrer est basée sur les résultats d'un algorithme fourni par chacun des noeuds du cluster. OpenMosix prend en compte les caractéristiques et les performances de chaque noeud (charge système, type et vitesse du processeur, taille mémoire...) et utilise ces informations pour décider de la migration des processus.
- un « *algorithme de dissémination d'information probabiliste » :* Chaque noeud envoie à intervalle régulier des informations sur ses ressources à un ensemble d'autres noeuds choisis au hasard.
- un « *algorithme de répartition de charge* » (« *load balancing* ») Celui ci tente en permanence de réduire les différences de charge entre les noeuds en migrant les processus sur d'autres noeuds L'algorithme est exécuté sur chaque noeud. Les différences sont calculées par paires de noeuds tirées au hasard.. Bien entendu la migration tient compte de l'état de l'occupation mémoire sur le noeud de destination. Il n'est pas question de migrer un processus sur un noeud dont la mémoire serait saturée. L'algorithme est conçu pour mettre le maximum de processus dans le plus grand espace mémoire pour éviter le swapping.
- un algorithme de « *gestion de la mémoire* » Pour contrebalancer une migration excessive de processus sur un noeud, l'état de la mémoire est surveillé de manière à éviter la saturation mémoire et le swapping des processus. L'algorithme est déclenché dès qu'un processus commence à paginer. Dans ce cas cet algorithme s'oppose à l'équilibrage de charge et tente de trouver un noeud avec une mémoire suffisante.

Le principal avantage de OpenMosix par rapport à d'autres types de cluster est donc sa capacité à répondre dynamiquement aux variations de ressources des différents noeuds et donc à des conditions d'exécution *irrégulières et non prédictibles*. Chaque noeud intégré au cluster calcule et communique en permanence sa charge système aux autres noeud, de telle manière que lorsqu'on lance un programme, le cluster décide sur quel noeud il va être exécuté.

On peut considérer OpenMosix comme un système intelligent de gestion de queue de processus : un « scheduleur intelligent ». A tout moment, dans un cluster OpenMosix un processus est donc assuré d'obtenir les meilleures conditions d'exécution. Il s'ensuit *in fine,* si les processus durent assez longtemps, une réduction globale importante du temps d'exécution des processus. Ainsi, tirer profit d'un cluster comme openMosix revient à écrire des applications qui lancent des processus. Si une application « *forke* » plusieurs processus fils, ceux ci seront candidats pour être automatiquement migrés vers le meilleur noeud. Il en résultera un gain global dans les temps de calcul.

OpenMosix donne l'illusion de transformer un ensemble de machines en réseau en une sorte de machine unique multiprocesseurs. Mais bien entendu il y a des différences. Dans une machine SMP les processeurs partagent une seule et même mémoire et les temps d'échange des données sont très rapides. Avec un cluster openMosix les temps d'échange entre chaque noeud seront limités par la vitesse du réseau. OpenMosix donne des performances tout à fait satisfaisantes sur un réseau commuté à 100Mb/s, mais un commutateur Ethernet et des cartes Gigabits amélioreront bien sûr l'efficacité du cluster en diminuant les temps de migration inter noeuds.

2.2 Mise en oeuvre de openMosix

La migration de processus et l'équilibrage de charge de openMosix est assurée uniquement par un ensemble d'algorithmes qui s'exécutent uniquement au niveau du noyau Linux. openMosix se présente donc simplement sous la forme d'une modification (« patch ») à appliquer au noyau Linux. Aussi, installer openMosix, c'est :

1. Modifier les sources du noyau Linux par le fichier de « *patch* » de openMosix
2. le recompiler pour obtenir un nouveau noyau intégrant les fonctionnalités openMosix

3.créer un fichier de configuration /etc/openMosix.map (ou lancer le daemon de découverte automatique)
4.installer les outils d'administration openMosix-tools et l'interface graphique openmosixview
5.modifier le secteur de démarrage (grub ou lilo) et démarrer la machine sur ce nouveau noyau.

C'est tout! Avec ce nouveau noyau openMosix, et un seul fichier de configuration (*/etc/openMosix.map*) définissant la liste des noeuds du cluster (voir plus bas), la machine qui démarre s'intègre automatiquement au cluster, et est prête à recevoir et envoyer des processus vers les autres noeuds. On peut alors vérifier par la commande « *mosmon* » ou « *showmap* » que les noeuds se voient entre eux, puis lancer quelques processus et vérifier qu'ils migrent bien vers les autres noeuds du cluster : Dans les quelques secondes suivant le lancement de plusieurs processus on voit avec les commandes « *mosmon* », « *mps* », « *mtop »,* que ceux-ci sont répartis sur les différents noeuds du cluster.

Chaque noeud du cluster openMosix est apte à calculer quelle est la meilleure machine voisine sur laquelle faire migrer. Il en résulte que dans un cluster de type openMosix, tous les noeuds sont *égaux en fonctionnalités*. Il n'y a pas de relations maître-esclave, et donc pas de notion de serveur « maître ». Chaque noeud peut envoyer et recevoir des processus de la part des autres. Toutefois par commodité nous parlerons de « noeud principal » pour désigner la machine sur laquelle se connectent les utilisateurs et à partir de laquelle seront lancés les processus. Cette machine est en général celle qui possède les comptes utilisateurs et le plus grand espace disque et donc un serveur de disque. On verra plus bas que OpenMosix a développé son propre service de distribution de fichiers via le réseau. Celui ci s'appelle oMFS (« *openMosix File System* », cf. plus bas) et remplace NFS moins adapté pour amener les données à l'endroit où se trouve le processus en cours d'exécution.

2.3 Installation de la distribution openMosix

On peut choisir d'installer openMosix sous forme de RPM précompilé ou bien à partir des fichiers sources. Installer openMosix revient uniquement à savoir compiler un noyau Linux et le lancer au démarrage. En tout premier lieu, il est donc nécessaire d'installer les sources du système Linux. Attention**,** openMosix ne fonctionnera correctement qu'avec un noyau Linux standard (mirroir de www.kernel.org), c'est à dire non modifié par une distribution comme Redhat, Mandrake, Suse ou autre. Pour l'installation de openMosix proprement dit, deux paquetages de base sont nécessaires :

- le fichier *openMosix* lui même, est en fait un fichier de différences, destiné à « patcher » le noyau Linux. Les patches openMosix suivent bien entendu les versions des noyaux Linux (openMosix 2.4.21-1.bz2..pour un noyau linux-2.4.21)
- le second paquetage nécessaire est *openMosix-Userlandtools* : Ce paquetage contient tous les exécutables nécessaires à l'administration et à l'exploitation du cluster en mode console. On y trouve les fichiers de configuration (*openMosix.map* et *openMosix.conf),* les scripts de lancement au démarrage, ainsi que les commandes */bin/mps* et */bin/mtop* commandes modifiées de « ps » et « top » incluant le numéro de noeud du cluster en regard des processus affichés. Enfin on trouve un daemon « *omdiscd* » qui est un programme qui effectue la découverte automatique des machines possédant un noyau openMosix, par diffusion « multicast ».

2.4 Configuration de openMosix : fichier *openMosix.map* ou découverte automatique

Avant de lancer openMosix il est nécessaire de configurer quelques fichiers. Au sein du réseau local, deux manières sont disponibles pour indiquer quelles machines feront partie du cluster :

- La première est d'utiliser le fichier */etc/openmosix.map*. Celui ci contient la liste des adresses IP des machines faisant partie du cluster. Chaque adresse IP est associée à un numéro de noeud.. Ce fichier *doit être identique* sur chaque noeud du cluster. Il s'agit là d'une configuration statique. Elle est utile dans le cas d'un cluster stable dont les noeuds ne changent pas. Lors du lancement de openMosix, ce fichier est lu et la communication réseau est initiée avec les noeuds déclarés. Si on enlève une machine ou qu'on en rajoute une nouvelle, elle ne sera pas prise en compte tant qu'on n'aura pas modifié ce fichier, et relancé openMosix sur chaque machine.

- La seconde manière pour configurer openMosix est d'utiliser un « daemon » de découverte automatique (*/sbin/omdiscd)* qui par diffusion multicast va permettre de découvrir les autres noeuds du réseau, s'en faire connaître, et les intégrer dans le cluster. Cette manière convient parfaitement pour un cluster dynamique dont le nombre de noeuds peut varier. C'est la manière que nous avons choisie lorsque nous rajoutons des noeuds dynamiquement avec LTSP+openMosix (cf plus bas). Ce daemon permet donc de s'affranchir d'une déclaration statique des machines du cluster dans le fichier *openmosix.map,* de la copie du fichier et du redémarrage de openMosix sur chaque noeud. Avec le daemon *omdiscd* chaque machine se déclare toute seule vis à vis des autres noeuds, sans intervention de l'administrateur système.

2.5 Types d'exploitation envisageables avec openMosix

On peut envisager plusieurs types d'exploitation de cluster. Considérons que nous avons un ensemble permanent initial de « n » serveurs openMosix. Nous pouvons alors rajouter de nouvelles stations de travail, appartenant à des utilisateurs, simplement en les faisant démarrer sur un noyau openMosix.

Grâce à la souplesse de paramétrage de openMosix, on peut alors constituer :

- un « cluster spécifique de serveurs » (« *server pool* ») : cas le plus simple où les stations de travail des utilisateurs ne font pas partie du cluster. Elles n'ont pas booté sur un noyau openMosix. Elles ne reçoivent donc pas de processus étrangers de la part des autres machines. Dans ce cas, il est nécessaire que le cluster aient un système de compte pour les utilisateurs afin de leur permettre de se connecter sur une des machines du cluster (ssh ou xdm par exemple) et lancer leur processus directement sur le cluster.

- Un « *cluster mixte* » (« *single pool* ») : dans ce cas nous faisons rentrer les stations de travail des utilisateurs qui ont démarré sur un noyau openMosix dans le cluster. Elles font alors partie intégrante du cluster et peuvent alors envoyer et recevoir des processus à d'autres noeuds. En paramétrant le système de fichier /proc/hpc spécifique au noyau openMosix on peut obtenir toute sorte de combinaison pour la migration des processus :
  - envoyer et recevoir des processus
  - en envoyer et ne pas en recevoir ou inversement, en recevoir et ne pas en envoyer

- Un « *cluster adaptatif* » troisième possibilité dans laquelle les stations de travail des utilisateurs se joignent au cluster pendant un certain temps puis le quittent. On peut imaginer tirer partie de certains ordinateurs personnels non utilisés pendant la nuit ou les congés. Un simple script lancé à une certaine heure permet à une machine personnelle de s'intégrer au cluster ou de le quitter.

2.6 Administration de openMosix : maîtriser la migration des processus

Un processus est lancé à partir d'un noeud d'origine et exploite certaines ressources du noeud local. Selon les ressources exploitées le processus pourra être migré ou pas. On a vu que par défaut, la migration des processus de openMosix s'effectue de manière automatique, dynamique et préemptive sans intervention de l'administrateur. Il n'en demeure pas moins que dans certains cas on souhaite pouvoir contrôler, s'opposer ou même provoquer cette migration. openMosix propose tout un ensemble d'outils permettant d'administrer le cluster et réguler la migration des processus.

Toute la configuration, le comportement des noeuds et l'administration de openMosix s'appuie sur le système de fichiers virtuel ***/proc/hpc.*** Ce système de fichiers est exploité par le biais de quelques commandes en mode ligne comme « *mosctl* ». On trouvera dans le openMosix HOWTO et le manuel de la commande « *man mosctl* » une liste détaillée des possibilités offertes par ce système de fichiers.

- Dans ***/proc/hpc/admin*** on contrôle le comportement de la migration de processus sur la machine locale, en plaçant des 1 ou 0 dans certains registres. La commande ***mosctl*** permet une meilleure ergonomie pour ne pas avoir à écrire 1 ou 0 directement dans les registres du file system */proc/hpc.* . A titre d'exemple :

`echo 1> /proc/hpc/admin/block`	`mosctl block`	empêche l'arrivée de processus étrangers, sur le noeud local, et comme « lstay » empêche aux processus locaux d'aller ailleurs.
`echo 1 > /proc/hpc/admin/bring`	`mosctl bring`	renvoie tous les processus migrés, sur le noeud d'origine
`echo 1 > /proc/hpc/admin/expel`	`mosctl expel`	i) renvoie les processus locaux sur leur noeud d'origine et ii) comme « block » interdit d'en recevoir de nouveaux. Intéressant pour sauver les processus avant de stopper Mosix ou rebooter
`echo 1 > /proc/hpc/admin/stay`	`mosctl stay | nostay`	interdit la migration automatique d'un processus en dehors du noeud
`echo 1 > /proc/hpc/admin/lstay`	`mosctl lstay | nolstay`	Interdit aux processus locaux de migrer ailleurs.. Mais les processus étrangers qui s'exécutent sur le noeud local, peuvent encore migrer ailleurs.

- Dans ***/proc/hpc/nodes*** on trouve un ensemble de registres permettant de connaître l'état des noeuds (vitesse du processeur, mémoire, charge CPU, état du noeud...). Là encore, la commande ***mosctl*** permet de lister les valeurs de ces registres pour un numéro de noeud donné.

- Dans ***/proc/<PID>***/ on trouve un ensemble de caractéristiques relatives aux processus en cours d'exécution de numéro <PID>. On peut savoir notamment sur quel noeud un processus s'exécute dans ***/proc/<PID>/where***, combien de fois il a migré dans ***/proc/<PID>/nmig***, s'il est bloqué dans ***/proc/<PID>/lock,*** et pourquoi il ne peut pas migrer dans ***/proc/<PID>/cantmove***

- Notons 2 autres registres très importants relatifs aux processus : ***/proc/self/lock*** et ***/proc/$$/lock*** vont permettre de contrôler la migration à partir d'un processus courant. Ainsi en plaçant 0 ou 1 dans ***/proc/$$/lock*** on permet, ou on interdit, la migration de processus « fils » lancés par ce processus courant.

Ces registres sont importants par le fait que les *processus fils héritent des conditions de migration* du processus père. Ainsi si le processus père (bash) ne permet pas la migration, alors aucun des processus fils ne sera migré, et eux mêmes ne permettront pas la migration à d'autres processus fils. Toutefois, en dépit de cet héritage, on peut forcer la migration d'un programme soit

- en mettant 1 dans /***proc/$$/lock*** (numéro du processus courant en cours d'exécution) ce qui autorisera le script fils à migrer vers d'autres noeuds.
- en mettant 1 dans /***proc/self/lock (***numéro de processus du shell courant)
- ou encore en lançant tout simplement un processus particulier avec la commande ***mosrun -l***

2.6.1 Les commandes mosctl , mosrun et migrate

Afin de contrôler la migration, on peut avec la commande « *mosrun* » forcer le noeud d'origine sur lequel se fera le lancement d'un processus. Un utilisateur peut par exemple être connecté sur le noeud 1 et en tapant ***$ mosrun -2 monprog.exe*** faire en sorte que le noeud d'origine du processus monprog.exe, soit le noeud numéro 2. L'option ***-h*** de ***mosrun -h prog.exe*** permet d'indiquer que le programme devra rester sur son noeud d'origine. Inversement l'option ***-l*** de ***mosrun -l prog.exe*** permet de forcer la migration de ce processus, même si le shell courant ne le permet pas. Enfin, La commande « ***migrate*** » permet à tout moment de forcer la migration d'un processus en cours d'exécution vers un autre noeud. Exemple *migrate <pid> 720* fera migrer le processus de numéro <pid> sur le noeud 720.

2.6.2 Précautions à prendre lors de la configuration d'un noeud du cluster

Si openMosix a les vertus de faire migrer des processus sur d'autres noeuds moins chargés, il n'est pas souhaitable de lui faire migrer n'importe quel processus! Par défaut openMosix ne fera pas la différence entre un processus de calcul, que lance un utilisateur et qui mérite d'aller s'exécuter plus rapidement ailleurs, et un processus système comme « *init* » ou « cups » qui, s'ils venaient à migrer sur d'autres noeuds entraîneraient de sérieux problèmes de performances. C'est dans ce contexte que l'on voit que openMosix est un cluster de calcul (HPC) destiné à certains processus de calcul et non pas un cluster de haute disponibilité (HA) destiné à de la redondance d'applications du système d'exploitation.

Pour ces raisons, il est nécessaire d'empêcher par défaut la migration de certains processus du système d'exploitation. Aussi la première chose à faire avant de mettre openMosix en exploitation est de modifier le fichier */etc/inittab* afin d'empêcher les processus fils du processus « *init* » de migrer! openMosix prévoit donc de faire précéder les commandes figurant dans */etc/inittab* par */bin/mosrun -h*. Cela assure que les processus systèmes initiaux lancés par « init » seront bloqués sur la machine locale. L'état « bloqué » des processus est bien entendu *transmis par héritage, à chacun des processus « fils »* que les processus initiaux vont engendrer. De la même façon lors de sessions graphiques sous KDE ou GNOME, j'ai trouvé avantage à bloquer dans */etc/inittab* les processus lancés par le Display Manager. Sans quoi on pouvait avoir certains processus fils comme *kdeinit* ou *DcopServer* qui migraient ailleurs, et posaient des problèmes. En conclusion, l'expérience montre qu'il est préférable de bloquer la majorité des processus du système d'exploitation, et de permettre de façon explicite la migration uniquement pour des processus de calcul par les commandes « *mosrun -l* ».

2.7 Ce qui migre et ce qui ne migre pas avec openMosix

On a vu que le domaine de prédilection pour exploiter au mieux le cluster OpenMosix est un ensemble d'applications qui lancent de multiples processus. On comprend aisément que l'on ne peut pas migrer n'importe quel processus. Certains utilisent des ressources physiques non partageables sur d'autres nœuds, et seront donc « bloqués » sur le noeud d'origine.

Actuellement la limitation la plus importante sur les clusters openMosix est celle portant sur l'utilisation de la « mémoire partagée » et des « threads ». Les applications qui utilisent de la mémoire partagée (par exemple Apache, Mathematica, MySQL, Python...) ou des threads ne peuvent pas être migrées. Les processus utilisant les threads n'ont pas d'espace mémoire propre. Ils utilisent l'espace mémoire du processus père de la machine où ils ont été créés. Il est donc en l'état actuel très difficile de les faire migrer sur d'autres machines. On trouvera une liste de quelques applications ne pouvant pas

bénéficier de la migration automatique de processus sur http://howto.ipng.be/openMosixWiki/index.php/don't. Ce sont par exemple les applications utilisant les appels systèmes *shmget()*, *shmat*(), *shmdt*() et *shmctl*() http://howto.ipng.be/openMosix-HOWTO/x1225.html. Ces applications seront donc exécutées localement sur le noeud d'origine. Ce qui n'empêchera nullement d'autres applications de bénéficier de la répartition de charge de openMosix.

- Une application qui « *forke* » est un bon processus qui permet à ses processus fils d'être exécuté sur d'autres noeuds, afin de répartir au mieux la charge au sein du cluster.
- Une application « monolithique » ne bénéficiera que peu ou pas de openMosix. Tout au plus sera t-elle envoyée sur un autre noeud présentant de meilleures ressources... elle ne saturera donc pas la machine si d'autres processus sont lancés.
- Une application dont le temps d'exécution est inférieur à quelques secondes ne sera jamais migrée. Par exemple, lors de la compilation du noyau Linux on s'attend toujours à ce qu'on puisse bénéficier de la migration de openMosix pour accélérer la compilation globale. Mais les compilations des différents programmes sources du noyau ne durent pas suffisamment longtemps pour être migrées (on utilisera dans ce cas ***distcc*** http://distcc.samba.org/).

Les applications de calcul avec un faible taux de communication inter processus tireront le meilleur profit de openMosix. Dans le cas contraire, si l'application est construite pour échanger de multiples données entre les processus, on aura avantage à utiliser PVM ou MPI dans les applications sur le cluster openMosix.

Tous les efforts de développement de openMosix sont actuellement concentrés sur la fonctionnalité de migration des applications utilisant de la mémoire partagée. Une équipe de développement de openMosix (maask team) travaille sur un patch permettant de faire migrer les processus utilisant de la mémoire partagée. Un patch est déjà disponible en beta-test mais n'est pas encore intégré dans le développement officiel de openMosix. On trouvera ce développement sur http://mcaserta.com/maask/

2.8 Le système de fichier de openMosix : oMFS et DFSA

Le problème lorsqu'un processus qui réalise des lectures/écritures sur disque, a été migré sur une autre machine, est de faire en sorte qu'il puisse accéder à ses données sur disque sans avoir à être renvoyé sur son noeud d'origine. Si un processus créé un fichier sur le disque d'un noeud A, comment pourra-t' il encore écrire sur ce fichier du noeud A s'il est migré ailleurs,?

oMFS (*OpenMosix File System*) est le système de fichiers de OpenMosix spécialement conçu pour résoudre ce problème d'accès aux fichiers distants. DFSA « *Direct File System Access* » est la couche logicielle qui permet d'intercepter les appels systèmes du processus en cours d'exécution sur un noeud distant.

- oMFS permet d'accéder au système de fichier des autres noeuds comme s'il était monté localement. C'est un système de fichiers dédié au cluster réseau qui comble certaines lacunes de NFS. oMFS est « symétrique » c'est à dire que chaque noeud voit et accède le système de fichiers des autres noeuds. Il n'y a pas de goulet d'étranglement qui pourrait exister avec un serveur de fichier unique centralisé. Quelle que soit la machine sur laquelle il a été migré, un processus est assuré d'y trouver un montage du système de fichier qu'il avait sur le noeud d'origine.
- DFSA utilise oMFS et permet d'intercepter les appels systèmes que fait un processus au système de fichier. Il exécute donc localement les appels systèmes comme *open, read, write....* sans que le processus ait à retourner sur son noeud d'origine. DFSA permet aux processus distants (ceux qui ont été migrés) d'effectuer localement (à l'endroit où ils ont été migrés) les appels systèmes sur le système de fichier plutôt que d'avoir à les renvoyer sur le noeud d'origine.

L'activation de oMFS et DFSA se fait lors de la compilation du noyau. Du point de vue de l'administrateur, l'exploitation est similaire à celle d'un système de fichier Unix. Il suffit de « monter » le système de fichier */mfs* comme un autre en rajoutant une ligne particulière dans */etc/fstab* pour le point de montage ***/mfs***. Il est utile d'activer les 2 options MFS+DFSA sur tous les noeuds, puisqu'elles améliorent les accès des processus aux fichiers sur l'ensemble des noeuds. Sans DFSA les processus devraient retourner sur le noeud d'origine pour effectuer leurs appels au système de fichier, ce qui peut devenir énormément pénalisant si le processus effectue beaucoup d'entrées/sorties. On aurait pu utiliser NFS pour partager les disques, mais NFS ne conserve pas de cache (cache consistency)... il peut donc y avoir des problèmes si 2 processus sur 2 noeuds distincts écrivent en même temps sur le même fichier. oMFS apporte des fonctionnalités de cohérence de cache indispensables à un partage du disque en cluster. oMFS assure que si plusieurs processus écrivent simultanément sur un seul fichier, l'intégrité des données et les informations du fichier seront préservées.

2.9 Les outils d'exploitation graphique : openMosixView

OpenMosix est un cluster de type SSI (*Single System Image*) ce qui signifie qu'on doit pouvoir administrer le cluster comme si c'était une seule machine SMP homogène. Aussi, à partir de n 'importe quel noeud, il faut pouvoir par exemple localiser des processus sur un noeud du cluster, les arrêter, les faire migrer sur d'autres noeuds, voir le status des processus, voir la charge CPU de chaque noeud du cluster, voir l'occupation mémoire de chaque noeud, etc... La suite openMosixView (http://www.openmosixview.com) est l'interface graphique permettant d'exploiter le système de registres */proc/hpc* de manière ergonomique. Elle comprend un ensemble d'applications graphiques permettant d'avoir un *état synoptique du cluster en temps réel*, et de lancer toutes sortes d'actions d'administration sur les processus en cours d'exécution. On peut ainsi consulter et gérer de façon interactive la migration des processus sur les noeuds, recueillir et afficher des statistiques d'utilisation de chaque machine du cluster. Nous citons ci après quelques fonctionnalités essentielles :

- **openmosixview** : Cette fenêtre principale permet de visualiser les informations de chaque noeud du cluster : adresse IP, numéro de noeud, nombre de processeurs, charge CPU, taux d'occupation mémoire, efficacité de la répartition de charge sur l'ensemble des noeuds, et un « poids » relatif à la puissance CPU de chaque noeud. Ce poids est calculé sur la base de plusieurs paramètres comme la puissance du processeur, cependant l'utilisateur peut modifier cette valeur par l'interface graphique (ou *mosctl setpeed*) afin de forcer l'utilisation d'un noeud.

- **openmosixprocs** : fenêtre de gestion des processus sur le noeud local. L'affichage est similaire à ce que produirait « *mtop*» mais avec des possibilités d'interaction sur les processus. On peut repérer (couleur verte) les processus « migrés » en cours d'exécution sur une autre machine. En cliquant dessus on peut agir sur les processus : les envoyer sur un meilleur noeud, les faire revenir sur le noeud d'origine, les arrêter...

- **openmosixcollector** : est le daemon permettant de collecter en temps réel des informations de charge sur les noeuds du cluster et de les stocker dans /tmp où ils pourront être exploités graphiquement par openmosixanalyser. **openmosixanalyzer** : permet d'analyser les données collectées par openmosixcollector et d'afficher une évolution temporelle de la charge système CPU et de l'occupation mémoire de chaque noeud.

- **openmosixmigmon** : permet de gérer la migration des processus sur une vue en 2 dimensions. Chaque noeud est représenté sous la forme d'un cercle autour des quels gravitent les processus. Cette fenêtre offre à l'utilisateur de grandes *possibilités d'interaction*. On peut glisser/déposer un processus sur un noeud avec la souris.

2.10 BenchMarks et tests de charges – applications de prédilection de openMosix

Les applications de prédilection tournant sur un cluster openMosix sont celles qui lancent beaucoup de processus de calcul. Pour illustrer pédagogiquement les performances de openMosix, nous avons reproduit le test de Ferri [10] consistant à faire de la compression audio MP3 à partir de fichiers .wav. Le but est de constituer des fichiers *.mp3* à partir de fichiers *.wav* extraits d'un CD, au moyen d'un programme d'encodage audio comme « *bladeenc* » (ou « *flac* »). Le test est intéressant car les processus *bladeenc* font de la compression de données, et donc beaucoup de calculs qui utilisent énormément le processeur et la mémoire. Les processus durent en outre un temps suffisant en mémoire pour bénéficier des algorithmes de migration et de répartition de charge de openMosix.

Le test consiste à faire une simple boucle pour lancer « n » processus de compression. Après quelques secondes, ceux ci sont migrés sur plusieurs noeuds libres du cluster. On compare les temps d'exécution obtenus, par rapport à la même boucle de programmes dans laquelle on aura bloqué volontairement la migration avec la directive *mosrun -h.*

Dans ce cas d'école, les résultats sont assez éloquents. Nous gagnons plus de 50% de temps d'exécution.

- En bloquant la migration (*mosrun -h*), c'est à dire avec des conditions d'exécution classiques sur un seul noeud, l'encodage .mp3 des sept fichiers .wav (donc 7 processus concurrentiels) du disque de test que j'ai choisi, dure 5 minutes environ.
- En autorisant la migration de OpenMosix (*mosrun -l*), les 7 processus se répartissent sur les différents noeuds du cluster et bénéficient des meilleures ressources mémoire et CPU. L'encodage mp3 réalisé sur des noeuds libres de toutes charge se termine en 2 minutes environ.

Si on veut estimer « *l'overhead* » c'est à dire la surcharge induite par la migration des processus d'un noeud à l'autre via le réseau, on peut lancer l'exécution des processus depuis le noeud 1 et les faire se bloquer sur le noeud 2 (mosrun -2 prog.exe &). Puis comparer avec une exécution locale sans migration sur le noeud 1. La différence ente les 2 tests est due au temps de migration entre les noeuds 1 et 2. Bien sur cela dépend de la taille des processus et de la bande passante du réseau, mais dans les conditions ci dessus l'overhead est évalué par Ferri[11] à environ 2-4% du temps total. On trouvera dans Gadaud [12] une autre étude sur les performances de Mosix.

2.11 Conclusions : avantages et inconvénients de openMosix

2.11.1 Avantages

- openMosix peut fonctionner avec une architecture matérielle hétérogène : plusieurs type de processeurs et mémoires
- openMosix donne l'illusion d'un seul ensemble de calcul efficace dans lequel on distribue les processus entre les noeuds du cluster. Pour un coût beaucoup plus faible, le cluster se comporte presque comme un SuperCalculateur.
- Par défaut, les interventions des utilisateurs et de l'administrateur sur le cluster sont extrêmement réduites. Le cluster régule sa charge automatiquement. L'utilisateur peut contrôler et agir sur les processus depuis un seul nœud.
- Avec openMosix, ll n'est pas nécessaire d'écrire des applications particulières et d'avoir des connaissances spéciales en programmation pour bénéficier de la migration de processus. Tout le travail d'éligibilité pour la migration des processus se fait automatiquement au niveau du noyau.
- La « granularité » de openMosix est le processus. Plus une application lance de processus, plus elle bénéficie de l'étalement de la charge et d'un gain performance global. Ainsi tirer profit d'un cluster comme Mosix revient à écrire des applications qui lancent des processus. Si une application « forke » plusieurs processus fils, ceux ci seront automatiquement migrés vers le meilleur noeud. Il en résultera un gain global si les processus durent suffisamment longtemps.
- L'installation et la configuration d'un cluster openMosix sont simples, surtout avec la découverte automatique des noeuds (*omdiscd*) par diffusion multicast
- L'interface graphique d'administration openmosixview est simple, efficace et permet d'avoir une vue synoptique du cluster comme si c'était une seule machine
- Une communauté openSource de développement très active. Les derniers développement portent sur :
  - une option de limite de charge permettant de spécifier jusqu'à quelle limite un noeud peut accepter des processus
  - un « masque » logique permet de spécifier un ensemble de noeuds vers lesquels on autorise les processus à migrer
  - la migration pour des applications utilisant des thread et de la mémoire partagée
  - une reprise à chaud des processus (checkpoint) dans le cas ou un noeud aurait dû être arrêté subitement.

2.11.2 Inconvénients

- Forte dépendance vis à vis du noyau Linux (openMosix ne tourne pas sous Windows).
- Les applications utilisant de la « mémoire partagée » ou des « *threads* » ne peuvent pas actuellement migrer au sein d'un cluster openMosix. Des développements sont en cours pour combler cette lacune.
- On n'aura pas de gain de performance si on lance un seul processus sur une machine du cluster, le processus ne se déplacera pas, sauf si openMosix trouve une meilleure machine. openMosix n'accélère pas l'exécution d'un processus unique lancé sur une seule machine. Une application monolithique, lancée comme un processus unique ne tirera aucun bénéfice d'un cluster openMosix. Tout au plus, et c'est déjà un gain, openMosix la fera migrer vers une machine présentant de meilleures conditions d'exécution (CPU plus puissant, charge moindre). En revanche, si cette même application est lancée plusieurs fois sous forme de plusieurs processus, ceux-ci seront alors distribués sur les autres noeuds du cluster, et il en résultera un gain de temps d'exécution global.
- Les processus s'exécutant très rapidement et qui ne durent pas suffisamment longtemps en mémoire, ne bénéficient pas des algorithmes de openMosix. Pour un processus durant quelques secondes il n'y aurait d'ailleurs aucun bénéfice à le faire puisque les temps de migration rivaliseraient avec les temps d'exécution
- Si aucune librairie de parallélisation n'est requise pour tirer partie de openMosix, en revanche il est préférable de bien concevoir son application et d'écrire des programmes très modulaires qui lancent des processus par *fork*() et *exec*().

3 LTSP : Linux Terminal Server Project

LTSP (http://www.ltsp.org/) a été créé par Jim McQuillan et son équipe en 1996. LTSP est une distribution tournant sur Linux, qui permet de mettre en place un « serveur de terminaux ». Ce serveur fournit par le réseau un système Linux (ou Windows) complet à des PC sans disque (« clients légers »). Au terme de la séquence de démarrage, les PC clients ont accès à un disque partagé, un système de fichier, un DNS, un service d'authentification et une connexion graphique sur le serveur, et ont donc accès à tous les logiciels installés sur le serveur. Les buts recherchés de LTSP sont d'avoir un système simple permettant de centraliser l'administration des PC clients sur un seul serveur, et de pouvoir utiliser des PC clients très bon marché, voire d'anciens PC recyclés qui ne satisfont plus dans le domaine des systèmes et suites bureautiques Microsoft actuelles.

Par défaut LTSP fournit un système Linux aux PC clients, cependant quelques distributions complémentaires comme « rdesktop » (http://www.rdesktop.org), ou « wilisystem » (http://www.wilisystem.com/), permettent au PC client de se connecter au serveur et obtenir le bureau d'un système Microsoft Windows TSE.

La distribution LTSP s'appuie sur quelques autres services Unix qu'il est nécessaire d'avoir mis en place au préalable :

- Un serveur DHCP permettant d'attribuer automatiquement une adresse IP aux clients et de leur proposer un noyau de démarrage
- un serveur TFTP: nécessaire pour permettre au PC client de télécharger le noyau LTSP/Linux
- un serveur NFS : nécessaire pour permettre au client de monter le système de fichier racine « / » de la distribution LTSP et les « *home directory* » des utilisateurs.
- un serveur NIS : nécessaire pour permettre l'authentification des utilisateurs du PC client, sur le serveur.
- un serveur XDM : serveur nécessaire pour permettre au PC client d'obtenir une bannière de connexion graphique (KDM, GDM) avec laquelle les utilisateurs entreront en session sur le serveur.

Chacun de ces services peut bien entendu se trouver sur n'importe quelle machine du parc. Il n'est pas nécessaire de réinstaller sur le serveur LTSP des services déjà en exploitation sur d'autres machines. La configuration LTSP est extrêmement souple et permet de désigner nommément l'adresse IP de chacun des serveurs à l'aide d'un seul fichier de configuration « *ltsp.conf* ». Comme le signalent Auroux et al. [2], il ne faut pas sous estimer le travail préalable d'administration, mais le serveur LTSP lui même est relativement simple à mettre en oeuvre dans un parc de machines possédant déjà un ensemble de services Unix basiques (DHCP, TFTP, NFS).

Les PC clients de LTSP sont davantage des « clients légers » plutôt que de simples « terminaux X » dans la mesure où LTSP permet de lancer un serveur XWindow sur les PC clients, (pour utiliser les applications graphiques tournant sur le serveur LTSP), mais aussi d'utiliser localement certains services déportés comme le lecteur de disquette et cdrom, une carte son, une imprimante connectée sur un port série ou USB... Grâce au paquetage « l*ocal_apps »* on pourra même faire tourner sur le client LTSP certaines applications comme Netscape et soulager ainsi le serveur LTSP central.

Dans quelles circonstances pouvons nous avoir besoin de LTSP?

1. si vous pensez qu'un grand nombre de PC (secrétariats, passagers...) utilisant le plus souvent une simple suite bureautique, du mail et un navigateur Web, n'ont pas besoin d'être des Pentium IV 2Ghz avec des disques de 80Go...
2. si vous avez un grand parc de PC à gérer et voulez alléger votre charge en réduisant le nombre de systèmes à administrer...
3. si vous chercher à centraliser davantage l'administration de vos systèmes à partir d'un seul
4. si vous appréciez les développements openSource et les préférez à des solutions commerciales parfois chères (Citrix)
5. si vous voulez en conclusion réduire les coûts d'administration, budgétaires et humains ... alors LTSP est fait pour vous !

LTSP permet donc d'utiliser très facilement et de réexploiter comme « clients légers » d'anciens PC ne répondant plus à des besoins de bureautique actuels. Pour exemple, sur un Pentium II avec 64Mo de RAM, une fois le système chargé, les disque montés par NFS et l'entrée en session faite au travers d'une interface graphique comme KDE ou GNOME, la *sensation* du point de vue de l'utilisateur est pratiquement identique à celle que l'on a si l'on travaille directement connecté sur la console maîtresse du serveur. Sur un réseau commuté à 100Mb/s, les performances clients/serveur LSTP sont *excellentes*. Auroux et al. [2] signalent que les performances de LTSP sont supérieures à celles de « clients légers » de constructeurs.

Le projet LTSP est extrêmement ouvert et montre une grande activité de développement. Il s'enrichit régulièrement de nouvelles fonctionnalités comme par exemple la faculté de faire démarrer des PC possédant une carte « wifi ». Pour résumer l'intérêt de la communauté, LTSP a été élu meilleur projet Open Source 2003 à la « Linux World Conference » de San Francisco en Août 2003.

3.1 Composition et Installation de LTSP

3.1.1 Les paquetages de base

Pour installer et faire tourner LTSP, 4 paquetages minimum sont nécessaires (formats rpm ou tar.gz):

ltsp_core-3.0.9 : contient le système de fichier racine « / » du client LTSP. Ce mini système de fichier Linux s'installe dans le répertoire */opt/ltsp/i386* lequel doit être exporté et accessible par NFS, afin que le client LTSP puisse le monter lors de sa séquence de boot. Le chemin d'accès à ce système de fichier doit en outre être donné comme paramètre au serveur dhcp (*option root-path de dhcpd.conf voir plus bas)* afin que le client qui démarre connaisse l'endroit où se situe son système de fichier racine.

ltsp_kernel-3.0.10 : contient le noyau de démarrage Linux spécialement modifié pour les clients LTSP. *(vmlinuz-2.4.21-ltsp-1)*. Celui ci est installé dans le répertoire */tftpboot/lts* afin d'être accessible par un serveur tftp. Ce répertoire sera donné

en paramètre dans la configuration de DHCP (option « *filename* » de dhcpd.conf (cf. plus bas)) afin d'indiquer au client qui démarre où il doit aller chercher son noyau de démarrage. Bien entendu c'est ce répertoire qu'il faut paramétrer pour être la racine du serveur tftp.

ltsp_x_core-3.0.4 : contient le serveur Xwindow Xfree-4 de LTSP. Notons que la configuration du serveur Xfree-4 est effectuée complètement automatiquement par LTSP. Dans l'hypothèse ou le serveur Xfree-4 ne gérerait pas un certain nombre de cartes vidéo, LTSP fournit alors un grand nombre de paquetages de serveur X spécifique à telle ou telle carte vidéo (ltsp_x336_svga-3.0.0, ltsp_x336_mach64-3.0.0..etc)

ltsp_x_fonts-3.0.0: les polices de caractères de LTSP dans le cas où on n'utilise pas de serveur de polices comme XFS.

3.1.2 Exécuter des applications sur les clients légers

Outre les 4 paquetages de base, les deux paquetages suivants sont intéressants pour alléger la charge du serveur, en faisant en sorte que certaines applications s'exécutent du coté du PC client

ltsp_local_apps-3.0.0-0 : est le paquetage nécessaire si l'on veut utiliser le PC LTSP comme un vrai « client léger » c'est à dire pour pouvoir lancer certaines applications qui s'exécuteront sur le PC client plutôt que sur le serveur. Le paquetage contient donc les librairies, un serveur *rshd*, et les fichiers d'autorisation nécessaires (*pam, tcp_wrapper*) pour qu'une application puisse s'exécuter localement sur le client LTSP, après qu'elle ait été lancée par « rsh » depuis le serveur où l'utilisateur est connecté.

ltsp_local_netscape-3.0.0-0 : est un exemple d'application configurée pour qu'elle puisse s'exécuter sur le client LTSP. Une fois connecté sur le serveur xdm, un utilisateur peut lancer netscape par « rsh » depuis le serveur pour qu'il s'exécute sur le « display » du PC client.

3.1.3 Construire son propre noyau LTSP pour intégrer openMosix

Enfin on peut trouver un paquetage extrêmement intéressant « ltsp_initrd_kit-3.0.10-i386.tgz » qui est nécessaire si l'on veut construire et configurer son propre noyau Linux-LTSP. Ce paquetage contient les scripts et le tout premier système de fichier racine initial que l'on va utiliser en « RAM disk » quand le PC client démarre.

Dans quel cas peut on avoir besoin de construire et personnaliser son propre noyau LTSP? Le cas le plus démonstratif et qui nous préoccupe tout particulièrement dans cet article est par exemple de rajouter les fonctionnalités de openMosix au noyau LTSP. Pour ce faire il suffit de patcher les sources du noyau Linux avec le fichier de patch de openMosix, et de le compiler au moyen du fichier de configuration de LTSP qui est donné dans le paquetage « ltsp_initrd_kit ».

3.2 Le fichier de configuration de LTSP

La configuration de LTSP est extrêmement souple et paramétrable au moyen d'un seul fichier de configuration *lts.conf* qui se situe dans */opt/ltsp/i386/etc*. Chaque PC client peut être déclaré dans ce fichier. Il permet de désigner explicitement tous les serveurs intervenant dans la phase de boot de chaque PC clients. Les paramètres du fichier « *lts.conf* » ne servent qu'à désigner les serveurs NFS, DNS, NIS, XDM, SYSLOG qu'utiliseront les PC clients. Dans ce fichier :

- Une section « *default* » donne les paramètres par défaut pour l'ensemble des clients qui bootent.
- Une section spécifique à chacun des PC clients qui démarrent, permet de modifier individuellement les paramètres par défaut. Pour chaque PC client, on est alors maître d'étaler la charge sur différents serveurs. On peut personnaliser le type de clavier, d'écran, le niveau de démarrage, l'utilisation d'applications du coté « client »... et même le lancement de openMosix sur le client.

3.3 Comment démarre un client LTSP par le réseau, de A à Z ?

Il importe de connaître les phases de démarrage du PC client, afin de mieux paramétrer les différents serveurs et de diagnostiquer les problèmes éventuels.

Pour faire démarrer le PC client par le réseau on a le choix entre 2 méthodes principales:

- soit démarrer sur une disquette, qui devra alors contenir un noyau spécial appelé EtherBoot (http://www.etherboot.org/)

- soit en utilisant les fonctionnalités PXE de la carte Ethernet. On trouvera dans Morris [11] un long plaidoyer en faveur des démarrages de PC par PXE.

Dans les 2 cas, il nous faut un premier noyau de démarrage « Etherboot » destiné à être logé dans la ROM de la carte Ethernet. La seule fonction de ce mini noyau Etherboot est d'initialiser la carte réseau (obtention d'une adresse DHCP), puis de télécharger un autre noyau disponible par *tftp* sur le réseau local afin de faire démarrer le PC client sans disque.

Le site « http://rom-o-matic.net/ » permet d'obtenir des noyaux Etherboot pour toute sorte de carte réseau et pour un média de démarrage particulier (disquette, PXE, lilo...). Par exemple, pour une carte 3c905c le noyau Etherboot destiné à être copié sur disquette est le fichier eb-5.0.11-3c905c-tpo.lzdsk. Dans le cas d'un noyau Etherboot destiné à être téléchargé par PXE (eb-5.0.11-3c905c-tpo.lzpxe), il faudra alors placer ce noyau Etherboot non plus sur une disquette mais à un emplacement où il sera accessible par la requête TFTP que va faire PXE.

Avec LTSP version 3, si on exploite ces 2 types de démarrage (disquette ou PXE), le serveur DHCP devra pouvoir faire la différence entre :
- une première requête PXE provenant de la carte réseau, et demandant à obtenir le premier noyau Etherboot, et
- une seconde requête provenant du noyau Etherboot lui-même, demandant à télécharger le noyau suivant

Pour cela, les paramètres à donner dans le serveur DHCP sont indiqués dans le « PXE Howto » (*http://ltsp.org/documentation/pxe.howto.html*). Il faut utiliser la variable interne « *vendor-class-identifier* » de DHCP. Lors de la requête émanant de PXE cette variable contient le mot clé « *PXEclient* », alors que dans le cas d'une requête DHCP émanant du noyau Etherboot cette option contient le mot clé « *Etherboot* ». Ainsi avec un test « if » dans dhcpd.conf, on peut faire en sorte que le serveur DHCP délivre soit un premier noyau de démarrage Etherboot pour la carte réseau, soit le second noyau Linux/LTSP pour la suite du boot du PC client.

Le serveur DHCP fournit au client qui démarre un ensemble de paramètres bien connus comme l'adresse IP, le masque de sous réseau, adresse de routeur et des DNS… etc Mais pour satisfaire les premières phases de démarrage des clients LTSP, il devra fournir en plus :
- le chemin d'accès du futur noyau à télécharger avec l'option « ***filename*** ». Le noyau donné en paramètre de « *filename* » devra se situer dans une arborescence connue du serveur TFTP.
- le chemin d'accès pour monter le système de fichiers racine « / » avec l'option « ***root-path*** »
- l'option « ***next-server*** » permet d'indiquer au client qui boote que le serveur TFTP est situé sur une autre machine que le serveur DHCP et qu'il faudra faire une requête tftp vers ce « prochain » serveur.
- En outre avec l'option ***group***{} on peut grouper certains PC partageant les mêmes caractéristiques (carte réseau identique par exemple)

A la fin de cette première phase, le PC client a obtenu une adresse IP et un code Etherboot logé en ROM de la carte Ethernet. La phase suivante est lancée par le noyau Etherboot : nouvelle requête aux serveur DHCP et requête TFTP pour obtenir le noyau définitif LTSP. Le noyau LTSP est alors chargé en RAM et on entre alors dans les phases d'un chargement classique d'un noyau Linux. Le système de fichiers racine « / » provenant de */opt/ltsp/i386* du serveur, est monté par NFS et un changement de racine est fait. (L'ancien « file system » / qui était en ramdisk est déplacé sur /oldroot.).

On continue par la séquence normale d'un boot linux avec le premier processus « *init* » qui va lancer tous les autres processus selon le niveau de démarrage 3 ou 5 indiqué pour chaque PC dans « lts.conf ». Si l'option « *local_apps* » est mise à « Y » dans le fichier « *lts.conf* », le client NIS *ypbind* sera lancé, et les « *home directory* » des utilisateurs seront montés localement depuis un serveur NFS. La séquence de démarrage se termine en lançant le script *rc.local* dans lequel l'administrateur pourra intervenir en plaçant des directives pour personnaliser son installation du PC client LTSP. C'est dans ce fichier « rc.local » que nous ferons certaines actions pour le lancement automatique de openMosix sur les PC clients. Si on a choisi un niveau de démarrage 5 dans le fichier lts.conf, LTSP termine en lançant automatiquement le serveur Xfree-4 sur le poste client qui reçoit alors une bannière de connexion XDM (KDM ou GDM). Il ne reste plus à l'utilisateur qu'à se connecter sur le serveur.

4 Déploiement de la grappe de PC openMosix avec LTSP

On a vu qu'intégrer un PC dans un cluster openMosix revient à le faire démarrer sur un noyau Linux modifié. On comprend donc que si on trouve un moyen de faire booter des PC sur ce noyau particulier mis à disposition sur un système de fichiers *tftp*, on peut faire entrer n'importe quel PC dans l'anneau de calcul.

Aussi 2 manières s'offrent à nous pour déployer le cluster openMosix:

- soit cloner des disques de PC avec des outils de clônage classiques. Nous avons utilisé partimage [http://www.partimage.org] ou UDPCast [http://udpcast.linux.lu/?power-to-the-people] comme systèmes openSource de clonage de disque.
- soit utiliser LTSP, puisque LTSP (avec le trio magique DHCP/TFTP/NFS) fournit un système sur un serveur pour des PC clients sans disque. Il nous suffit de rajouter les fonctionnalités openMosix à celles de LTSP pour obtenir un noyau polyvalent, et de mettre ce noyau à disposition sur le serveur *tftp*.

Dans le second cas, nous avons utilisé la distribution « *ltsp-openMosix »* de W. Daniau [13]. Cette distribution astucieuse se propose juste de mettre à disposition, au bon endroit :

1.un noyau spécialement conçu pour LTSP, auquel on a rajouté les fonctionnalités de openMosix. Ce noyau est placé dans */tftpboot/lts*

2.les modules de ce noyau qui sont placés dans */opt/ltsp/i386/lib/modules*

Ensuite, grâce à DHCP et TFTP on peut faire démarrer un PC client sur ce noyau particulier possédant les fonctionnalités LTSP + openMosix. La distribution de W.Daniau est prête à l'emploi, elle s'installe automatiquement aux bons emplacements prévus pour LTSP et TFTP. En outre, elle rajoute une nouvelle option « *MOSIX = [Y|N] »* dans le fichier de configuration de LTSP « *lts.conf »* afin de choisir de lancer ou non le démarrage de openMosix au terme du boot d'un PC donné. Cette option est traitée dans le script *rc.local* à la fin du démarrage du PC.

Cependant, pour des raisons de maîtrise totale des options du noyau, nous avons choisi de construire notre propre noyau LTSP+openMosix afin de bien contrôler la constitution du noyau de A à Z. Il suffit pour cela de patcher les sources d'un noyau Linux avec le patch openMosix, et de le compiler avec le fichier de configuration de LTSP. La seule différence par rapport à un noyau openMosix vu précédemment est que le démarrage de ce nouveau noyau se fait par le réseau. Une fois la compilation du nouveau noyau LTSP+OpenMosix achevée, il est donc nécessaire de modifier le noyau par l'utilitaire « *mknbi* » (qui signifie « *make net bootable image* »), afin que ce noyau soit « *bootable* » par le réseau. Dans le répertoire *ltsp_initrd_kit*, on trouvera l'utilitaire « *buildk* » qui effectue automatiquement ce travail, et qui « taggue » le noyau avec mknbi.

Il suffit alors de copier le nouveau noyau (ltsp+openMosix « *taggué* » avec mknbi) dans le système de fichier de /*tftpboot* afin que celui ci soit accessible par *tftp*. et de copier les modules dont le noyau aura besoin, dans le système de fichier de ltsp. Pour finir on indique dans le fichier */etc/dhcpd.conf* le nouveau nom du noyau sur lequel les clients doivent démarrer (option « *filename* »). Les PC clients peuvent alors démarrer sur le nouveau noyau qui possède les fonctionnalités LTSP plus celles de openMosix.

Au terme du démarrage, non seulement le PC « client léger » obtient la bannière de connexion XDM d'un serveur quelconque, mais en plus il est intégré au cluster openMosix. Il partage son processeur et peut envoyer, recevoir et exécuter des processus.

En conclusion, on a créé un noyau openMosix et on le fait servir par LTSP. On met ainsi en place un moyen souple de faire démarrer n'importe quel PC du Laboratoire et de l'intégrer dans l'anneau de calcul sans aucune intervention sur le PC. Une fois l'administration des différents services faite (dhcp, tftp, nfs, xdm), il ne reste qu'a intégrer le PC client dans *lts.conf* afin de lui dire avec quels serveurs il doit démarrer.

4.1 déploiement sous forme de mini distribution : clusterKnoppix

Outre le procédé que nous avons testé et qui a retenu notre préférence, il est a noter l'existence d'une distribution « prête à l'emploi » sous forme d'une image ISO à graver. ClusterKnoppix http://bofh.be/clusterknoppix/ est une distribution Knoppix à laquelle on a intégré un noyau openMosix. En démarrant des PC sur le CD on construit le cluster : installation et administration sont réduites à néant !

5 Conclusions de l'étude

Avec openMosix nous avons trouvé un moyen de constituer un anneau de calcul de manière simple pour l'administrateur et efficace pour les besoins de développement et de calcul de notre laboratoire.

Avec openMosix nous évitons:

- l'achat de grosses machines de calcul parallèles qui sont très chères, et surtout
- l'achat de multiples machines individuelles pour chaque chercheur entraînant un surcoût important, et une grande charge d'administration

Avec openMosix nous réalisons:

- Un anneau de calcul dans lequel les processus sont échangés entre les machines de manière préemptive et automatique, à la recherche des meilleures conditions d'exécution sur un noeud présentant la charge la plus faible et la meilleure puissance.
- Une collaboration des PC du cluster entre eux : Les PC échangent en permanence leurs états de charge et de saturation mémoire, de manière à constituer une sorte de calculateur géant. Par défaut et sauf cas particuliers, les utilisateurs et l'administrateur n'interviennent jamais dans l'administration des processus.
- Une répartition de charge permanente. OpenMosix est une bonne solution pour étaler la charge de processus de calcul entre plusieurs machines et gagner globalement du temps d'exécution
- une centralisation et donc une économie budgétaire et d'administration !

OpenMosix *n'est cependant pas la solution ultime et la réponse à tous les problèmes de calcul*. La nature hétérogène des besoins de calcul dans les logiciels scientifiques *demande que l'on considère différentes solutions de cluster pour différents problèmes de calcul* (Bar et al. [4]). Selon le cas, on devra considérer également l'utilisation d'autres solutions à base de librairies PVM et MPI, ou se tourner vers d'autres types de systèmes de « clustering » comme OpenSSI http://www.openssi.org/ ou le projet CLIC de Mandrake [http://clic.mandrakesoft.com/index-en.html] ou encore RockCluster [http://www.rocksclusters.org/Rocks/]. Enfin, pour ne pas être déçu, il est bon de se rappeler que openMosix est un cluster de type HPC, qui répond uniquement à des besoins de calcul. Ce n'est pas un cluster de haute disponiblité (HA) prévoyant de la redondance de services. En l'état actuel il ne faut pas espérer de openMosix qu'il réponde à de l'étalement de charge de serveurs (Web, Mail ou autres)

Avec LTSP nous *distribuons* facilement un système linux à des PC sans disque. Nous réalisons un déploiement facile de nombreux PC pour lesquels nous n'avons plus à administrer les systèmes. Nous réduisons les coûts d'achat et d'administration à l'extrême. Avec quelques paquetages supplémentaires le client léger LTSP peut également se connecter à un serveur Windows TSE par RDP.

Enfin avec le couple LTSP+OpenMosix nous réalisons un déploiement facile du cluster dans la mesure ou dès le démarrage avec LTSP les PC obtiennent un noyau spécifique openMosix prêt à l'emploi qui leur permet d'être intégrés dans le cluster openMosix. Avec la souplesse de configuration qui permet de désigner nommément chaque serveur, nous étendons ou faisons varier à volonté la taille du cluster. L'extension du cluster peut être faite dynamiquement en cours d'exécution des processus... Des PC inutilisés peuvent être intégrés au cluster par un réveil automatique.

6 Références

[1]Milhaud G., Pagé O., Petit manuel anti-dépression à l'usage des administrateurs système et réseaux. *Actes du Congrès JRES2001*, pp. 213-226

[2]Auroux D. et al., Clients légers. *Actes du Congrès JRES2001*, pp. 521-528.

[3]PVM : Parralel Virtual Machine (*http://www.csm.ornl.gov/pvm/pvm_home.html*)

[4]Bar M., et al., openMosix vs Beowulf : a case study, Août 2002

[5]Le Cannelier St Paul C., Mosix : *Linux Magazine*, Septembre 2000, pp 46-50.

[6]Robbins D., Advantages of OpenMosix on IBM xSeries : http://www-1.ibm.com/servers/esdd/articles/openMosix.html

[7]Barak A., La'adan O., The Mosix multicomputer operating system for high performance cluster computing

[8]Bar M.,: OpenMosix Internals http://sourceforge.net/docman/display_doc.php?docid=10390&group_id=46729

*[9]*Buytaert K., *The OpenMosix HOWTO* : http://howto.ipng.be/openMosix-HOWTO

*[10]*Ferri R., The Secrets of openMosix. *Sys Admin The journal of Unix and Linux system administrators, Août 2003* article disponible sur "http://www.samag.com/documents/s=8817/sam0313a/0313a.htm et *(http://www-1.ibm.com/servers/esdd/articles/openMosixpart3.html)*

[11]Morris F., Réveil de machines à travers le réseau et démarrage à partir d'une image située sur un serveur, des outils utiles pour l'administration de parc. *Actes du Congrès JRES2001*, pp. 537-545

[12]Gadaud F., Mosix : http://www.gadaud.org/fabrice/experience/mosix.pdf. Rapport Ecole Polyechnique Université » de Nantes.

[13]Daniau W., LTSP-OpenMosix noyau de démarrage LTSP+OpenMosix: http://www.lpmo.edu/~daniau/ltsp-mosix