Cluster de PCs

sous GNU/Linux avec openMOSIX

Rapport de travail d'étude et de recherche

http ://openmosix.sourceforge.net/

Benoît Dejean & Paul Goergler

Maîtrise Informatique 2004
Directeurs de travail d'étude et de recherche :
Bruno Jobard & Laurent Lacayrelle

Département
d'Informatique

Université de Pau et
des Pays de l'Adour

Table des matières

1 Introduction **5**

2 Cluster **6**
2.1 Définition 6
2.2 Historique 6
2.3 Nécessité des clusters et impératifs 6
2.4 Conception 8
2.4.1 SSI 8
2.4.2 NUMA 8
2.5 Différents types de cluster 9
2.5.1 Clusters à hautes performances 9
2.5.2 Clusters de stockage 10
2.5.3 Clusters Haute Disponibilité 10
2.5.4 Clusters à répartition de charge 12
2.6 Exemples de clusters 13
2.6.1 Exemple détaillé : Terrascale 13
2.6.2 distcc 14
2.6.3 MPI : Message Passing Interface 15
2.6.4 Beowulf 15
2.6.5 Systèmes de fichiers pour cluster 16

3 openMOSIX **19**
3.1 Historique 19
3.2 Principe 20
3.2.1 Aperçu 20
3.2.2 Patch pour le noyau Linux 20
3.2.3 Facteurs limitant la migration 22
3.3 API 22
3.3.1 /proc/hpc 22
3.3.2 /mfs 24
3.3.3 libmos 25
3.4 Configuration 25
3.4.1 Le réseau 25

3.4.2 Le noyau . . . 25
3.4.3 Le daemon . . . 26
3.5 Outils openMOSIX . . . 26
3.5.1 Les outils en ligne de commande . . . 26
3.5.2 La suite openMOSIXview . . . 27

4 Expérimentations **30**
4.1 Configurations . . . 30
4.2 Traitements parallèles sur un même fichier : pipeline . . . 31
4.2.1 Protocole . . . 31
4.2.2 Observations . . . 31
4.2.3 Interprétation . . . 33
4.3 Rendu d'image . . . 33
4.3.1 Protocole . . . 33
4.3.2 Observations . . . 33
4.3.3 Interprétation . . . 35
4.4 Compression de pistes audio . . . 35
4.4.1 Protocole . . . 35
4.4.2 Observations . . . 36
4.4.3 Interprétation . . . 36
4.4.4 Extrapolation . . . 37
4.5 Utilisation normale . . . 37
4.6 Conclusions des bancs d'essai . . . 37
4.7 Aspect financier . . . 37

5 Conclusion **39**

Au sujet de ce rapport

Ce rapport a été rédigé en LaTeX en utilisant les logiciels Emacs ainsi que Texify pour l'inclusion de code source. Ces logiciels sont disponibles gratuitement et sont publiés sous licence GNU GPL [1, 2].

Licence

Chapitre 1

Introduction

De nos jours, de plus en plus d'applications demandent de plus en plus ressources : traitement d'images ou vidéo, applications scientifiques, base de données... Ces applications demandent énormément de temps machine et nécessitent beaucoup de mémoire. Traditionnellement, ce genre de tâche incombe à des super-calculateurs tels que ceux qui ont fait la renommée de fabriquants comme IBM, SGI ou Sun. Les super-calculateurs concentrent en une seule machine toute la puissance et la capacité de stockage nécessaire au traitement de l'information. Ils sont pour cela excessivement chers et peu extensibles.

Le clustering, c'est-à-dire l'élaboration d'une ferme d'ordinateurs ou cluster, permet de répondre aux mêmes besoins mais à des coups bien moindre. Il s'agit simplement d'agréger autant de machines que nécessaires afin de réaliser le traitement. Les machines sont des micro-ordinateurs tels que des PC ou des Macintosh et présentent donc un coup modeste : les machines qui composent un cluster ont les mêmes caractéristiques qu'une machine de bureau classique, telle que l'utilisateur lambda la connaît. L'agrégation de plusieurs machines permet également un meilleur rendement : une machine peut être oisive et sa puissance est alors inexploitée. Les clusters fournissent des solutions afin de mieux exploiter ces machines oisives en répartissant les tâches sur les différentes machines.

Objectifs

Ce travail d'étude et de recherche présentera les différents types d'applications pouvant bénéficier du clustering et les différentes solutions existantes. Nous nous intéresserons plus particulièrement au cluster de PC openMOSIX[1] : après une étude détaillée, nous présenterons nos expérimentations et leurs résultats.

[1] http ://openmosix.sourceforge.net/

Chapitre 2

Cluster

2.1 Définition

Un cluster est un agrégat homogène ou hétérogène de micro-ordinateurs localement reliés par un réseau informatique qui travaillent ensemble comme un ordinateur parallèle. Les clusters sont également appelés « ferme ». Chaque ordinateur composant un cluster est appelé « nœud ».

On distingue principalement 4 topologies de cluster :

- Maître-esclaves.
- à 2 nœuds.
- multi-nœuds.
- massivement parallèles.

2.2 Historique

Le clustering a été originellement développé par DEC dans les années 1980. Les solutions alors mises en place, en plus de permettre le calcul parallèle, permettaient déjà de partager des systèmes de fichiers ainsi que des périphériques. Ces premiers clusters devaient assurer à l'utilisateur la possibilité de lancer n'importe quel programme dans de bonnes conditions, c'est-à-dire avec des performances acceptables et de manière fiable.

2.3 Nécessité des clusters et impératifs

Comme nous l'avons déjà évoqué, le clustering apporte une solution innovante pour les applications nécessitant une grande puissance de calcul. Il apparaît aujourd'hui clairement que les machines, super-calculateurs et multi-processeurs, des fabriquants de matériels sont excessivement chers.

La conception des clusters est la résultante de la maîtrise de plusieurs facteurs essentiels.

Le coût par nœud : à performances brutes équivalentes, un ordinateur disponible dans le commerce se révèle 2 à 3 fois moins cher qu'une machine vendue par un constructeur comme Sun.

La puissance par nœud : la puissance que déploie aujourd'hui un micro-ordinateur est suffisamment colossale pour qu'on puisse l'exploiter dans un calcul.

La taille d'un nœud : la miniaturisation des composants et la création de nouveaux standards de matériels permet de réaliser des unités centrales de taille très réduite, faciles à aligner sur des étagères. L'encombrement est donc considérablement réduit : plusieurs dizaines de machines peuvent être stockées par m^3.

Le dégagement thermique d'un nœud : le refroidissement est l'un des plus grand enjeux des cluster. En effet, les architectures micro-ordinateurs utilisées pour la réalisation des clusters sont certes bon marché, mais ont un très important dégagement thermique. D'autant plus que dans un cluster, les machines sont nombreuses et concentrées. Cependant, avec des technologies modernes, telles que le refroidissement par eau ou azote liquide, le dégagement thermique est désormais mieux maîtrisé et atteint les performances des solutions des grands fabricants.

La vitesse d'inter-connexion : la rapide évolution des réseaux permet la réalisation des clusters possibles. Les super-calculateurs sont composés d'unités de calculs qui communiquent de manière extrêmement rapide par des bus. Sur un cluster, les unités de calculs communiquent par le réseau qui est infiniment plus lent, et beaucoup moins réactif qu'un bus interne. Mais la popularisation des équipements réseaux GigaBits et la baisse de leur coût, fournit une qualité de réseau suffisante au déploiement efficace d'un cluster.

La bonne conception d'un cluster réside dans la bonne équation de ces 5 variables par rapport à la puissance fournie.

Les clusters sont, par unité de calcul, bien moins performants que les solutions matérielles des constructeurs, mais pour un coût très inférieur, il est possible de rassembler un grand nombre de machines afin d'obtenir la puissance désirée. La conception d'un cluster est une tâche simple, du moins, bien moins complexe et coûteuse que la conception d'un super-calculateur.

La puissance des micro-ordinateurs allant croissant selon la loi de Moore, et leur prix allant décroissant, la puissance d'un cluster ne dépend finalement que d'un paramètre numérique : le nombre de nœuds. Bien sur les problèmes que posent une grande concentration de machines existent. Toutes les solutions destinées aux entreprises doivent impérativement être fiables, hautement disponibles et fournir des performances proportionnelles à la taille du cluster.

2.4 Conception

Cette section décrit différents types de conception de cluster et fait référence à différents types de clusters et applications citées dans la section suivante. Les rapports tels que nous les connaissons se lisent de manière séquentielle, mais les deux sections suivantes sont à lire en parallèle. Pour une compréhension, n'hésitez pas à les relire.

2.4.1 SSI

Le concept de *Single-System Image* (SSI) repose sur l'idée que les ressources d'un cluster (ou de tout autre système qui regroupent des ressources) doivent présenter une interface unique à l'utilisateur. L'utilisateur n'a pas besoin de connaître la réalité physique du système, c'est-à-dire en connaître la multiplicité ou connaître l'adresse d'un composant du système. Le système doit apparaître comme une seule entité cohérente. Dans un cluster SSI, il n'y a pas de nœud maître. L'architecture des SSI permet d'intégrer des matériels hétérogènes, le système gérant les différences et s'ajustant automatiquement. L'énorme avantage des SSI sur les mécanismes classiques comme MPI, est que les programmes n'ont pas besoin d'être réécrits.

Les SSI sont très souples parce qu'ils sont facilement extensibles (ajout de machines au cluster). Ils sont également relativement fiable car la multiplicité des machines permet d'assurer la continuité des services. Les SSI sont également beaucoup plus faciles à administrer.

Les SSI sont généralement utilisés pour réaliser des clusters à répartitions de charges et/ou de haute disponibilité et/ou à hautes performances.

openMOSIX que nous avons étudier en étudier en détail est un système de cluster SSI.

2.4.2 NUMA

Non-Uniform Memory Access (NUMA) est une architecture de système de mémoire. Cette architecture est utilisée dans les systèmes multi-processeurs et dans certains clusters. Dans de tels systèmes, un processeur doit pouvoir accéder à tout l'espace mémoire, indépendamment de son emplacement. Bien évidemment, l'accès à une mémoire locale est infiniment plus rapide que l'accès à une mémoire distante (c'est-à-dire une mémoire partagée entre plusieurs processeurs).

L'architecture NUMA est la couche logique qui permet cette accès uniforme. Cette architecture est mise en œuvre de manière matérielle dans les systèmes *Symmetric Multiprocessing* (SMP) mais également de manière logicielle dans les systèmes de clustering. Bien entendu, la vitesse de communication inter-nœuds au sein d'un cluster est sans commune mesure avec les performances obtenues sur un système SMP.

Cohérence du cache

De nos jours, la grande majorité des processeurs sur le modèle de von Neumann utilisent en plus de leur mémoire vive, une mémoire additionnelle de faible capacité à accès ultra-rapide, appelée cache. Cette mémoire doit être à chaque instant maintenue dans un état de cohérence vis-à-vis de la mémoire vive. Dans un système NUMA, les mêmes contraintes s'appliquent ce qui impliquent un certain surcoût : maintenir la cohérence de plusieurs caches par rapport à une seule mémoire physique partagée est loin d'être aisé. Certains systèmes NUMA n'assurent pas cette cohérence, en contre-partie de quoi, ils sont très difficilement programmables. Afin de permettre cette accès cohérent à prix abordable en terme de performance, les *cache-coherent NUMA* (ccNUMA) utilisent des matériels dédiés. Cependant, lorsque plusieurs processeurs procèdent à des accès répétitifs à la même zone de mémoire partagée, de tels systèmes sont mis à mal. La meilleure façon de réaliser la cohérence du cache sur une architecture implique donc également une certaine coopération du système d'exploitation. L'allocateur mémoire ainsi que le répartiteur (*dispatcher*) doivent allouer les processeurs et la mémoire de manière coordonnée avec le matériel qui assure la cohérence. Un bon système d'exploitation NUMA doit donc utiliser des algorithmes compatibles pour les tâches d'ordonnancement et de verrouillage de ressources.

2.5 Différents types de cluster

Ces 4 catégories de clusters décrites ci-dessous ne s'excluent pas mutuellement. Beaucoup de systèmes sont créés pour répondre à plusieurs contraintes : par exemple, un cluster à hautes performances (pouvant compter plusieurs dizaines/centaines de machines) est humainement inexploitable, il est donc couplé à une technologie de répartition de charges.

2.5.1 Clusters à hautes performances

Le but de ce type de cluster est de pouvoir rivaliser, en termes de performances, avec les super-calculateurs classiques.

Pour arriver à se rapprocher de la vitesse des échanges ayant lieu au sein d'un ordinateur mais avec une connexion Ethernet, les protocoles de communication doivent être très performants sans pour autant surcharger le réseau. Les différents nœuds étant reliés par un LAN, les pertes de paquets sont quasi-nulles. Le protocole *Transfert Control Protocol* (TCP) [18] basé sur *Internet Protocol* (IP) [19] génère cependant beaucoup trop de paquets superflus, tels que des accusés de réception, ce qui peut encombrer inutilement le réseau, induire un surcoût non-négligeable et entraîner une baisse des performances. Le protocole TCP/IP est en cela trop générique pour être satisfaisant, mais d'autres protocoles « plus léger » (faible surcoût) existent,

notamment *Message Passing Interface* (MPI) qui est une *Application Program Interface* (API) pour les développeurs d'applications de calculs parallèles. Cette couche applicative assure des communications optimales au sein d'un cluster. L'entreprise Bull propose des clusters linux avec une API MPI développé part Scali[1] nommé MPI Connect[2]

Ce type de cluster peut remplacer un super-calculateur dans beaucoup d'applications, notamment celles qui sont par nature, décomposables :

- mathématique pour résoudre des problèmes complexes dans des applications telles que le calcul matriciel.
- génétique : étude du génome.
- géo-physique : modélisation et analyse de phénomènes.
- chimie : par exemple pliage de molécules.
- météorologie : simulation climatique à l'échelle planétaire.

Ce type de cluster, également appelé « Cluster à haute-performance », est également employé, par exemple, en infographie, dans le but de modéliser des scènes en 3D de plus en plus complexes et « photo-réalistes ». Les studios Pixar[4], célèbres pour leur films d'animation (le plus récent étant « Le Monde de Nemo »[5]). Des outils spécifiques pour des réalisations de moindre envergure mettent aussi à disposition des solutions de calculs parallèles : Lightwave[6] ou Maya[7].

2.5.2 Clusters de stockage

Contrairement aux clusters scientifiques, avec ce type de cluster, ce n'est pas la puissance de calcul qui est recherchée, mais une grande capacité de stockage.

Les entreprises utilisent de plus en plus des applications demandant de « gros » flux de données nécessitant un espace de stockage de plus en plus conséquent et dépassant souvent la capacité de quelques disques durs classiques. Ce système permet d'avoir un vaste espace de stockage virtuel. La distribution des données permet également des entrées/sorties nettement plus rapides.

Ce type de cluster peut se « combiner » avec les clusters scientifiques permettant à l'utilisateur de travailler avec des fichiers de grandes tailles et de bénéficier d'une puissance de calculs pour les traiter, le tout en minimisant les transferts sur le réseau.

2.5.3 Clusters Haute Disponibilité

Les systèmes offrant des services de plus en plus critiques, ils sont de plus en plus vulnérables aux failles tant matérielles (problème au niveau du

[1] http ://www.scali.com/

[2] http ://www.scali.com/index.php ?loc=18

réseau, surchauffe, dysfonctionnement) que logicielles (application interrompue). Les clusters dits de haute disponibilité sont là pour rendre ces systèmes tolérants aux pannes.

Grâce à la redondance des services, un tel système est quasiment invulnérable aux pannes. Ceci est particulièrement nécessaire dans les environnements à risque où une défaillance du système informatique peut être critique, voir mettre en danger la pérennité de l'information ou pire, menacer des vies humaines. Ce type de cluster est également utilisé par les entreprises commerciales afin d'assurer une continuité de service à leurs clients : le temps d'inactivité induit des pertes financières très conséquentes (pénalité, baisse de productivité, site web marchand inaccessible, etc) et décrédibilise l'entreprise.

Dans ce type de système, on utilise un service nommé *FailOver Service* (FOS). Ce service permet la gestion des services (démarrage/arrêt) mais seulement pour 2 ordinateurs. Le nœud primaire (maître) et le nœud secondaire (esclave) sont reliés via une connexion très peu sujette aux défaillances comme une connexion par port série, ainsi le nœud maître ne risque pas d'être remplacé par son esclave si il s'agit d'un problème de gestion du réseaux ou de connectique. En effet le nœud maître (actif) envoie un signal appelé « battement de cœur », ou *heartbeat*, à son nœud esclave. (Figure 2.1)

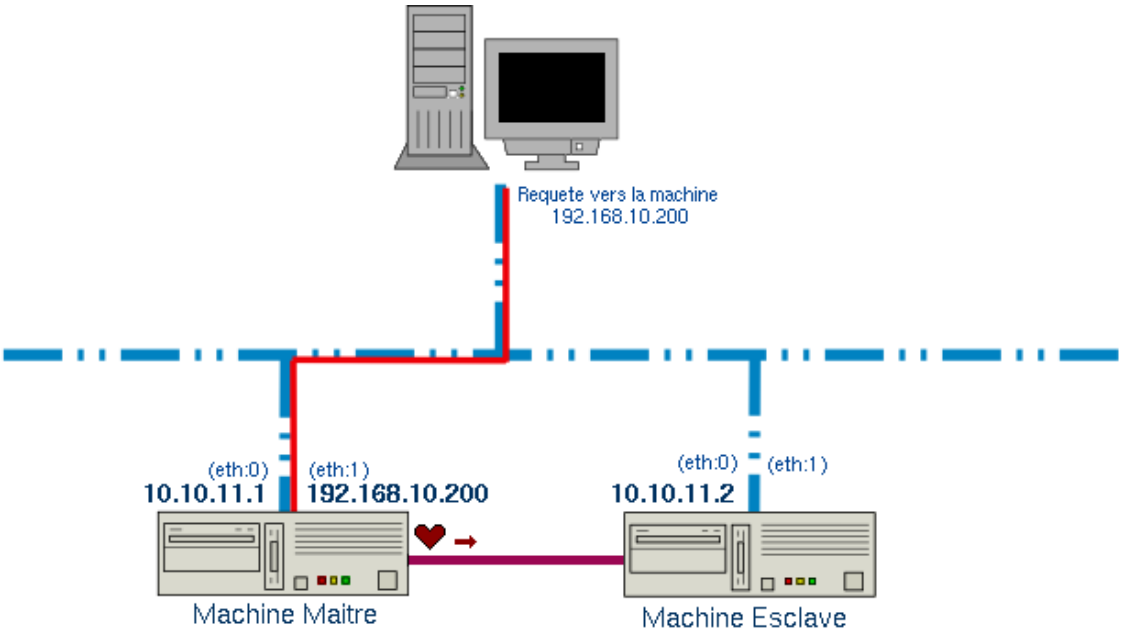

FIG. 2.1 – Fonctionnement normal

Dès que le nœud esclave ne reçoit plus le « battement de cœur » du nœud primaire, il le déclare comme « mort », ne faisant plus partie du cluster et

prend son identité complète. Ce remplacement d'identité est rendu possible grâce à l'*IP Aliasing*. L'*IP ALiasing* permet de définir une interface réseau avec plusieurs adresses IP différentes.

Grâce à tout ceci, si un nœud venait à avoir une défaillance, il serait immédiatement remplacé par son nœud secondaire. Le nœud secondaire en « sommeil » étant une image exacte de son nœud maître. (Figure 2.2)

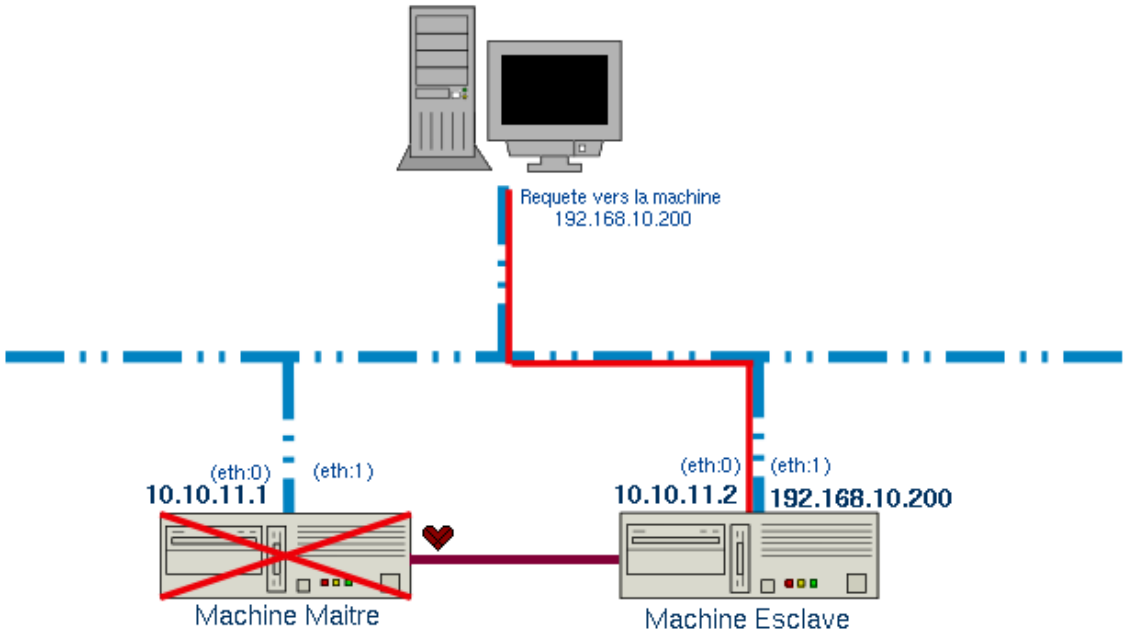

FIG. 2.2 – Scénario de panne et prise relai

L'utilisateur ne ressentira aucun désagrément si un nœud primaire devient indisponible et la continuité du service sera assurée.

2.5.4 Clusters à répartition de charge

Les systèmes à répartition de charge permettent de distribuer l'exécution de processus systèmes ou réseaux à travers les nœuds du cluster.

Cela fonctionne sur le principe du client/serveur. Le nœud serveur à comme tâche de réceptionner le processus et de l'envoyer sur le nœud le moins chargé ainsi elle peut traiter quasiment instantanément le processus qui lui a été attribué. Il est aussi possible de spécialiser des nœuds pour certains traitements.

Ce type de cluster peut être composé d'un parc hétérogène (systèmes d'exploitations différents mais aussi matériels différents). Ceci est rendu possible grâce aux systèmes client/serveur, le serveur contrôlant les charges des différents systèmes joue le rôle d'intermédiaire entre les différents nœuds.

Cette architecture a nécessité l'utilisation de protocoles de communication spécifiques permettant cette distribution efficace de la charge.

Ce type de cluster est surtout largement utilisé dans le domaine du réseau et plus particulièrement sur les services lourds comme les serveurs Web ou les serveurs de fichiers. Ce type de serveur permet aussi une répartition efficace des ressources matérielles.

2.6 Exemples de clusters

Les solutions de clustering les plus populaires sont basés sur les systèmes d'exploitation Unix, tel que GNU/Linux ou FreeBSD, combinés aux technologies propriétaires telles que MOSIX[3] ou libres comme Beowulf[4] et bien sur openMOSIX. Sun Microsystems[5] propose également une solution de clustering appelée « Grid Engine ».

2.6.1 Exemple détaillé : Terrascale

Le System X « Terrascale » de l'université de Virginie [6] États-Unis, le troisième ordinateur le plus puissant au monde en Novembre 2003, est un cluster composé de 1100 Apple[7] Macintosh G5 (machine bi-processeur) et fonctionne grâce au système Mac OS X et à son noyau Mach [8]. Ceux-ci totalisent :

- 2 x 1100 processeurs PPC 970 cadencé à 2GHz
- 4Go x 1100 de mémoire vives, soit 4,3 To
- 160Go x 1100 de capacité de stockage, soit 171,2 To.
- 3MW de consommation électrique.
- un système de refroidissement de plus de 2 millions de BTU équivalent à un vent au sol de 100Km/h.

Le réseau inter-connectant l'ensemble des ordinateurs est composé de 96 commutateurs. Chaque machine est connectée grâce à un lien 20Gbps pour une latence réseau minimale. Ce système a été conçu pour en prouver la faisabilité et pour répondre aux applications scientifiques en tous genres. Les outils tels que le compilateur gcc et l'API MPI sont utilisés. Le coût estimé de ce cluster est de $5,2 millions, ce qui en fait selon les experts, le super-calculateur au meilleur rapport qualité/prix jamais construit.

[3] http ://www.mosix.org
[4] http ://www.beowulf.org
[5] http ://www.sun.com
[6] http ://computing.vt.edu/
[7] http ://www.apple.com

2.6.2 distcc

distcc [9] est un programme qui distribue des tâches de compilation sur les différentes machines d'un réseau. distcc n'est pas un SSI car l'utilisateur doit renseigner les différentes machines acceptant des tâches. distcc a un fonctionnement simple : la machine maître envoie à une machine esclave un fichier source (unité de traduction) ; l'esclave la compile avec son compilateur localement disponibles, puis renvoie le code objet à la machine maître. distcc est très souple pour ces raisons. Il permet de répartir efficacement le travail de compilation entre les machines, et ce avec un surcoût négligeable. Contrairement à bon nombre d'autres systèmes de clustering, distcc se satisfait de réseau à faible débit, tel que des LAN 10Mbps : en effet, du fait de son fonctionnement, les communications entre les différents processus mis en action se résume à l'envoi de fichier source et à la réception de code objet. Ces transferts sont minimes et sont petits devant le temps de compilation d'une unité. Cependant, afin de toujours améliorer la qualité du service, distcc peut effectuer ces transferts de fichier en mode compressé grâce à l'algorithme LZO [10] (meilleur ratio compression/temps).

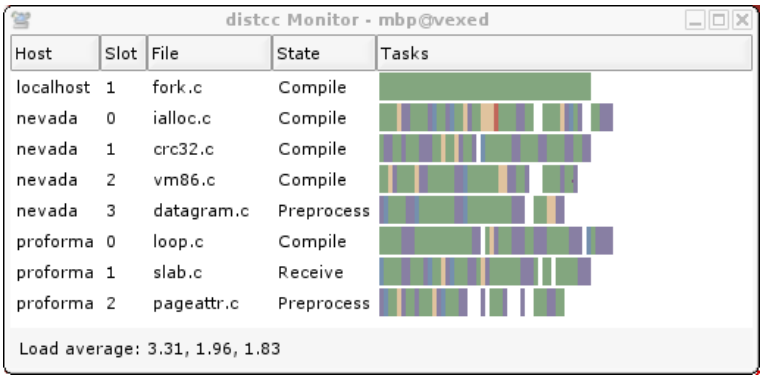

FIG. 2.3 – distcc : moniteur d'exécution, les tâches de compilation sont distribuées

À contrario, un SSI muni d'un système de fichiers partagé à un fonctionnement plus simple : le programme utilisateur n'a pas à gérer les transferts de fichiers et la gestion des processus. L'utilisateur se contente de lancer une tâche de compilation, le système se charge alors de répartir les tâches s'il juge cela nécessaire et fournit de manière transparente l'accès aux fichiers. Ceci est complètement transparent pour l'utilisateur, qui n'a même pas conscience du phénomène.

distcc vise une catégorie de tâches très spécifiques et remplit sa tâche de manière optimale. Sa mise en place est très simple et il s'adapte à tous les systèmes.

2.6.3 MPI : Message Passing Interface

MPI Forum [21] est un consortium créé en 1989 regroupant une soixantaine de personnes venant d'une quarantaine d'organisations Européennes et Américaines comme les laboratoires universitaires et industriels ainsi que les principaux fournisseurs de machines parallèles. Le but de ce consortium est de définir un standard c'est-à-dire une norme pour l'échange de messages inter-processus sur des architectures parallèles. Pour cela, une interface de programmation a été définie de manière à être pratique, portable, efficace, souple et performante.

Au début, MPI était destinée aux architectures multi-processeurs à mémoire distribuée (chaque processeur ayant sa propre mémoire) et aux clusters de machines mais également aux architectures mixtes (cluster de machines SMP). Bien que la gestion des machines multi-processeurs à mémoire partagées , des architectures MIMD[8] et SIMD[9] n'est pas explicite (pas de support explicite pour les threads), elle reste possible.

La première implémentation de cette norme fut achevée en 1992, la dernière version (2.0) a publiée en 1995. Cependant, on peut constater que son intégration dans des langages de dernière génération tel que Python, Perl (etc) a été réalisée sous forme de bibliothèques de fonctions avec un fort couplage avec le langage support (utilisation des mécanismes de sérialization natifs). Tandis que dans les langages plus « classiques », l'insertion de directives MPI se fait souvent sous forme de macros/directives pré-processeur.

MPI est l'API la plus utilisée par les scientifiques (mathématiciens, chimistes, physiciens...) car MPI a été avant tout créée pour leurs besoins spécifiques. Comme on le constate, MPI ou PVM *Parallel Virtual Machine* [22] ne sont pas pour tout le monde, c'est pourquoi on a conçu des systèmes de clustering ne nécessitant pas de réécriture de programmes ou de connaissances poussées en matière de parallélisation d'algorithmes.

2.6.4 Beowulf

Le projet Beowulf[10], démarré en 1994 par *National Aeronautics and Space Administration* (NASA), a été conçu comme un cluster à hautes performances utilisant des micro-ordinateurs peu chers. Ce n'est pas un logiciel à proprement dit, mais la coordination de plusieurs logiciels qui font un cluster Beowulf. Le système Beowulf fonctionne sur des ordinateurs utilisant des systèmes comme GNU/Linux ou FreeBSD, reliés par un réseau TCP/IP. Des bibliothèques, telles que MPI et PVM, doivent être installés sur chaque nœud afin de permettre aux ressources d'être partagées. Un cluster Beowulf est organisé autour d'un nœud central (contrôleur) qui gère l'ensemble des

[8] Multiple Instructions Multiple Data
[9] Single Instruction Multiple Data
[10] Poème épique de l'Angleterre médiéval

nœuds. La défaillance du contrôleur résulte en l'arrêt complet du cluster. Il faut noter que la réécriture des applications afin de tirer parti du cluster peut représenter un coût extrêmement important. C'est sur ces deux points que Beowulf n'est pas un SSI.

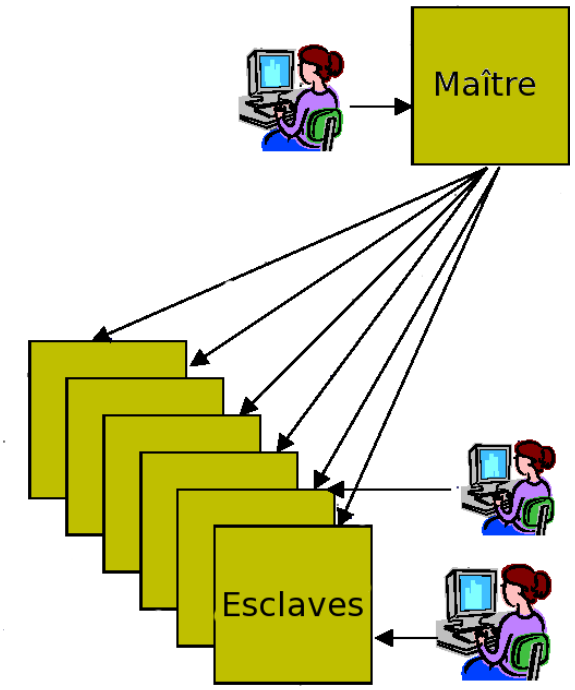

FIG. 2.4 – Beowulf

Cependant, il existe des solutions commerciales [11] clef en main Beowulf, cette solution de clustering étant très populaires et ayant fait ses preuves.

2.6.5 Systèmes de fichiers pour cluster

Voici une description de plusieurs systèmes de fichiers distribués couramment utilisés avec le système GNU/Linux.

NFS Network File System [11] n'est pas à proprement dit un système de fichier de cluster mais un système de fichiers réseau. Il permet de monter localement, c'est-à-dire définir un point d'accès, un système de fichiers distant. NFS est implémenté au niveau du noyau et fonctionne sous forme de daemon. En tant que système de fichiers réseau, NFS n'assure pas la cohérence du cache, ce qui peut avoir des conséquences en cas de détérioration du cluster, par exemple une déconnexion réseau. NFS fournit cependant une solution simple, efficace (pour des tâches

[11] http ://www.paracel.com/pc/hpc-services.htm

classiques) et standard : il est donc largement utilisé. NFS utilise RPC *Remote Procedure Call* ce qui peut se révéler cependant pénalisant dans des applications fortement liées aux Entrées/Sorties, à cause d'un certain surcoût inhérent.

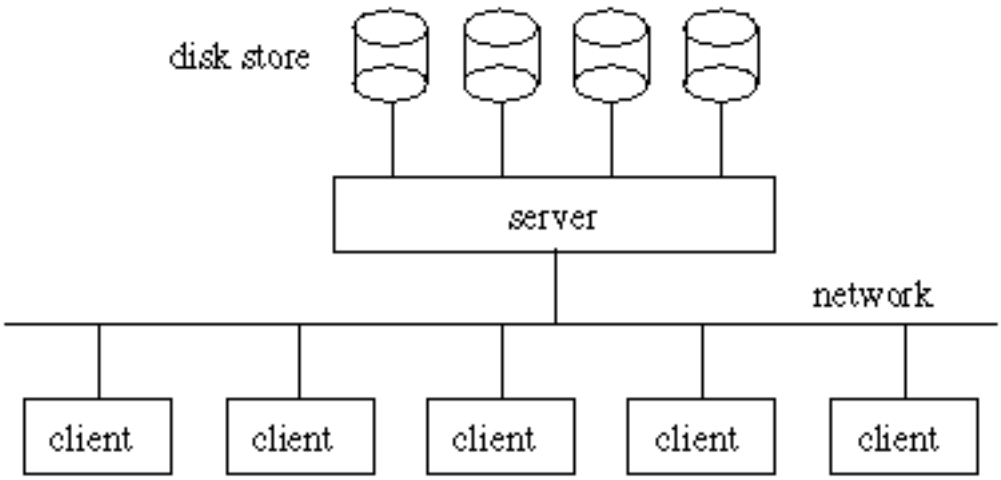

FIG. 2.5 – NFS permet aux clients d'accéder à des fichiers sans se soucier de leur emplacement physique.

oMFS openMOSIX File System est le système de fichiers virtuel d'openMOSIX qui permet un accès uniforme et un référentiel lexical des noms de fichiers (pas de collisions) aux systèmes de fichiers d'un cluster openMOSIX. Grâce à cela, l'utilisateur peut accéder, s'il en a les droits nécessaires, aux fichiers du nœud N. OMFS repose sur openMOSIX et cela lui permet de garantir la cohérence du cache en fonction des nœuds. OMFS n'est pas basé sur RPC mais est comparable à NFS.

PVFS Le Parallel Virtual File System [12] est un projet GPL qui tends à fournir un système de fichiers performant pour les clusters de PC. Le référentiel lexicographique est cohérent, les données sont physiquement distribuées sur le cluster, et les performances pour les applications utilisateurs sont optimales. Il s'agit donc d'un véritable système de fichiers parallèle qui répartit efficacement les Entrées/Sorties sur les différents nœuds du cluster. PVFS est composé de daemons, peut être optionnellement intégré au noyau, et fournit une API compatible avec MPI. PVFS s'intercale entre le système de fichiers local et l'utilisateur, ce qui le rends à la fois transparent et performant. La répartition des données sur le cluster permet des bien meilleures performances et évite les goulots d'étranglement du réseau.

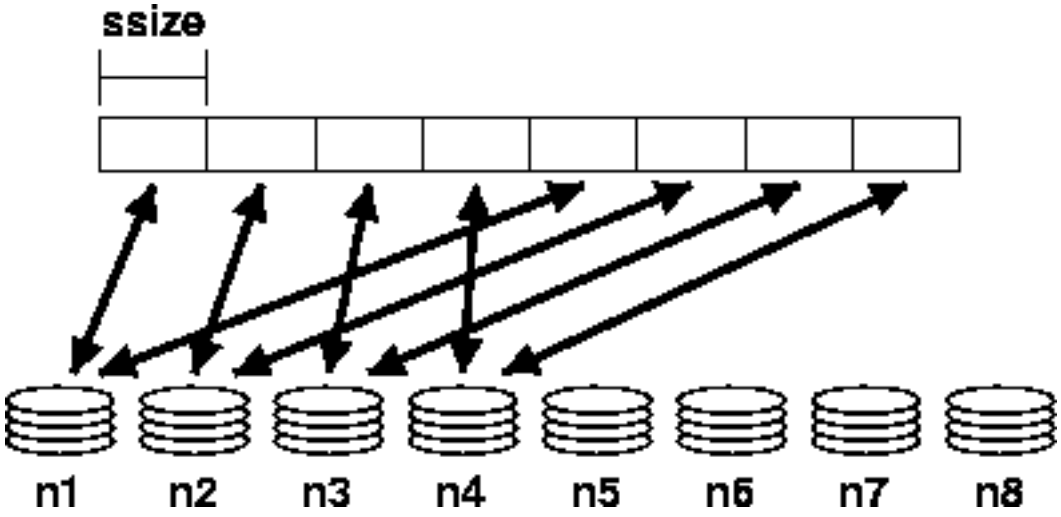

FIG. 2.6 – PVFS segmente un fichier qui se retrouve alors réparti sur plusieurs unités de stockage différente.

Chapitre 3

openMOSIX

3.1 Historique

openMOSIX est un projet dirigé par Moshe Bar[1], titulaire de doctorats en mathématiques et informatique, enseignant à l'université de Tel Aviv et à l'agence atomique des Nations Unie. Issu d'un fork du projet MOSIX, le projet openMOSIX a été initié pour pallier les problèmes de licences : openMOSIX est libre, toutes les parties du code sous licence propriétaire ont été remplacées. openMOSIX est donc une solution totalement libre sous licence GPL [2]. Le projet est hébergé par SourceForge [20]

openMOSIX a depuis évolué pour devenir une plateforme de clustering très complète. Depuis le développement de la branche openMOSIX, de nombreux clusters MOSIX ont basculé sur openMOSIX. L'ouverture complète du code source et les libertés garanties par la GPL ont fortement accéléré le développement du projet. Aujourd'hui, l'équipe de développement se compose d'une dizaine de personnes secondées par une dizaine de contributeurs extérieurs.

Outre le changement de licence, openMOSIX diffère de MOSIX sur les points suivants :

- un portage sur User Mode Linux (UML) [16] a été réalisé. UML est un dispositif fiable et sécurisé permettant d'exécuter plusieurs noyaux Linux (différentes versions, différentes options) dans l'espace utilisateur (user mode). Ceci permet de disposer de machines virtuelles disposant de leur configuration et de leur espace de stockage propre. Si une machine virtuelle plante, le système hôte reste lui disponible. UML permet également un contrôle très fin de droits, par exemple pour exécuter des services : ce portage montre bien la vocation serveur d'openMOSIX.
- le code du noyau gérant la migration des processus a été réécrit et est désormais plus propre et plus performant.

[1] Interview sur LinuxJournal : http ://www.linuxjournal.com/article.php ?sid=7007

- la latence du noyau a nettement diminuée. Sur MOSIX, les commutations *user-mode / kernel-mode* était très coûteuses.
- support de nouvelles architectures en plus de i386 : AMD IA64 et Dolphin. openMOSIX tend à fonctionner sur plus d'architectures (dont PPC).
- l'installation est très grandement simplifiée : des paquetages RPM, Debian et Slackware sont disponibles.
- l'ajout de documentation : la dimension libre d'openMOSIX a succitée de nombreuses contributions. Un wiki (système de rédaction collaboratif) permet de rassembler la documentation et les retours des utilisateurs.

3.2 Principe

3.2.1 Aperçu

openMOSIX est un SSI comme nous l'avons précédemment défini. openMOSIX est un cluster à répartition de charge, c'est-à-dire qu'en fonction de la charge de chaque nœud, le système va être capable de faire « migrer » des processus : un tâche lancée localement va pouvoir être exécutée par une machine distante et ce de manière transparente à l'utilisateur.

Ses particularités qui l'ont rendu populaires sont :

- sa publication sous licence GPL, avec tous les bénéfices pratiques et financiers que cela implique.
- sa bonne intégration au noyau Linux, système aujourd'hui relativement bien répandus.
- son système de fichiers oMFS
- sa boîte à outils très complète : une API complète et simple ainsi que des outils utilisateurs afin de configurer/administrer/surveiller le cluster.

3.2.2 Patch pour le noyau Linux

openMOSIX est un patch pour le noyau Linux, c'est-à-dire des modifications et des ajouts au code source de Linux, qui est un noyau sous licence GPL, dont les sources sont donc accessibles et disponibles. Ce patch représente environ 180.000 lignes touchant environ 200 fichiers. Ce patch interagit de façon subtil avec Linux : un petit nombre de modifications permettent de détourner le fonctionnement normal de la gestion du cpu `arch/i386` ainsi que l'unité de gestion de la mémoire `mm/` et biensur le coeur `kernel/`. L'algorithme de migration de processus et la gestion des ressources sont implémentés dans `hpc/` et constituent l'apport majeur du patch. Au niveau de oMFS et de la gestion des systèmes fichiers est également réalisé en `fs/` et `fs/mfs/`. openMOSIX doit également interagir avec les communications ré-

seaux on trouve donc également des modifications dans `net/`. Le patch est réalisé de manière subtile dans le sens où il ne remplace le coeur du noyau, c'est-à-dire qui capture les appels systèmes l'intéressant et les détourne : ainsi l'ordonnancement est réalisé et un programme exécutant un appel système dialoguera correctement avec son hôte.

Tous les appels systèmes ne sont pas supportés et certaines directives POSIX ne sont pas encore respectées.

Les points chauds de la réalisation d'openMOSIX sont la gestion de la mémoire ainsi que l'accès aux fichiers (sans passer par oMFS, pour charger un exécutable et ses ressources).

En effet, la gestion de la mémoire est critique car la moindre instruction est susceptible de déclencher des algorithmes simples sur une machine locale, mais difficiles à gérer dans le cadre d'une mémoire distribuée. Par exemple, l'accroissement de la pile provoque un appel implicite à `mremap` afin de redimensionner le segment de pile. Ceci est problématique : openMOSIX procède donc selon la politique de l'autruche et ne garantie l'exécution que dans un cadre applicatif classique. Dans le cadre d'une application utilisant de manière intensive la pile, jusqu'à sa limite, les développeurs conseillent de dimensionner soi même la pile avec une taille initiale afin de prévenir tout appel à `mremap`. Ceci est relativement lourd à gérer.

De plus les processus utilisant les mécanismes de mémoire partagée (*shm*) ne peuvent pas être migrés : en effet ce genre de mécanisme demande un arbitrage du noyau. On comprends donc que les différents processus attachés à un même segment de mémoire partagée doivent se trouver sur le même nœud. Il existe néanmoins un patch permettant la migration de ces processus : tous les processus attachés à un même segment de mémoire partagée sont alors migrés sur un même nœud. L'expérimentation réalisées par les auteurs de ce patch montre un gain de performance nul. En effet les processus communiquant par mémoire partagée sont souvent par nature peu aptes à la migration : il s'agit souvent de daemon, il subsiste donc un grand nombre de communications avec le nœud originel.

Quant à l'accès aux fichiers, *Direct File System Access* ou DFSA il est également limité pour les mêmes raisons : par exemple, l'appel système `mmap` ne peut exécuter que des projections anonymes (les projections partagées ne sont pour l'instant pas supportées pour le surcoût qu'elle implique sur un système à mémoire distribuée et systèmes de fichiers répartis). Les verrous sur fichiers sont également restreints.

En cas de panne d'un nœud, il existe un mécanisme de ramasse-miettes qui a pour tâche de rendre le cluster à son état normal en réparant les dommages causés par la déconnexion brutale. Cependant, cette procédure automatique peut prendre plus d'une heure pour que les nœuds toujours en activité abandonnent définitivement la connexion : cette durée est bloquante car elle empêche le démontage du système de fichiers perdu. openMOSIX a donc une tolérance aux pannes mais ce n'est pas sa priorité.

openMOSIX est un patch qui suit la philosophie de Linux : le bon sens. Les mécanismes difficiles à réaliser et peu utilisés sont laisser de côté pour se concentrer en priorité sur les tâches de bases qui sont exécutées en permanences.

Pour beaucoup plus de détails techniques : `Documentation/`

3.2.3 Facteurs limitant la migration

Certains facteurs sont à en prendre en compte : openMOSIX n'est pas en mesure de faire migrer tous les processus et s'interdit d'en faire migrer d'autres pour des raisons de performances. openMOSIX s'abstient de faire migrer les processus fortement liés aux Entrées/Sorties (que ce soient sur disque dur ou dans le cadre de communications IPC) et les processus à courte durée d'exécution ou faible consommation en temps CPU. Le mécanisme de migration a un coût, il est petit devant un travail de durée raisonnable, mais trop important pour des petits processus (tels que `ls, mv, top`, etc). Les expérimentations le montrent bien.

3.3 API

openMOSIX dispose d'une API complète qui permet de d'administrer, de configurer et d'obtenir des informations sur l'état du cluster.

3.3.1 /proc/hpc

Ceci s'effectue par l'interface `/proc` qui est monté à l'amorçage du noyau : on peut alors changer facilement certains paramètres concernant le nœud local et obtenir des informations statistiques sur les autres nœuds du cluster. Cette interface est très pratique car très dénudée : elle permet une administration très aisée.

openMOSIX fournit toutes les informations concernant tous les nœuds via le répertoire `/proc/hpc`. Toute modification (modification de la configuration ou récupération d'information) est répercutée à intervalle fixe (lequel est également paramétrable).

Cette interface permet à chaque nœud de connaître l'état du cluster afin d'effectuer la répartition de charge selon l'algorithme inventé par Moshe Bar : le système est donc décentralisé puisque chaque nœud prend la décision de faire migrer ses processus vers les nœuds de son choix. Ceci est un point fort d'openMOSIX, puisque ce système s'adapte linéairement à la taille du cluster (nombre de nœuds).

Usage

Chaque paramètre est défini par un fichier virtuel dans l'interface `/proc/hpc/admin`.

L'usage est très simple, par exemple :
- lecture du paramètre
  `cat /proc/hpc/admin/llimitmode`
- écriture du paramètre (modification)
  `echo 1 > /proc/hpc/admin/llimitmode`

Description

Voici une liste non-exhaustive des informations disponibles. La multiplicité des informations et la finesses des réglages qu'elles permettent montrent bien la qualité d'openMOSIX. Cette API, disponible sous deux formes – interface `/proc/hpc` et bibliothèque C – permet une utilisation avec n'importe quel langage support : l'interface `/proc/hpc` sera utilisé par des langages tels que sh ou php, libmos par des programmes C ou C++. Il est également possible de réaliser des ponts afin de disposer de libmos dans n'importe quel autre langage de programmation. Cette API est très riche et offre de nombreux moyens de réaliser des programmes/scripts d'administration du cluster. Cette API permet de reprendre la main sur l'auto-migration et d'ajuster à ses besoins la migration des processus.

Informations locales Ces informations permettent de configurer openMOSIX.

Fichier virtuel	Description
/proc/hpc/admin/loadcpulimit	Fonctionnalité récente qui permet de restreindre la charge imposée à un nœud. Permet de spécifier le temps CPU alloué aux processus arrivants et donc de contrôler de manière fine la charge d'un nœud ainsi que de forcer la migration pour imposer une forte charge à un nœud distant.
/proc/hpc/admin/loadlimit	Permet la limitation de charge.
/proc/hpc/admin/llimitmode	Restriction de la charge.
/proc/hpc/admin/cpulimit	Pourcentage de partage du CPU.
/proc/hpc/admin/speed	Vitesse relative (référence : Intel Pentium3 1 GHz).
/proc/hpc/admin/stay	Migration automatique des processus.
/proc/hpc/admin/lstay	Les processus locaux ne doivent pas migrés.
/proc/hpc/admin/block	Empêche l'arrivée de processus.
/proc/hpc/admin/expel	Renvoie les processus arrivants.
/proc/hpc/admin/bring	Ramène les processus migrés.
/proc/hpc/admin/decayinterval	Interval de collecte des informations.

Informations distantes L'interface par laquelle le nœud hote est informé de la configuration des autres nœuds du cluster.

Fichier virtuel	Description
/proc/hpc/nodes/<ID>/cpus	Nombres de CPU.
/proc/hpc/nodes/<ID>/load	Charge openMOSIX.
/proc/hpc/nodes/<ID>/mem	Mémoire disponible selon openMosix.
/proc/hpc/nodes/<ID>/rmem	Mémoire disponible selon Linux.
/proc/hpc/nodes/<ID>/speed	Vitesse relative.
/proc/hpc/nodes/<ID>/status	État du nœud.

Processus locaux et distants Paramètrage et informations sur les processus locaux et distants.

Fichier virtuel	Description
/proc/<PID>/cantmove	Pourquoi le processus ne peut être migré.
/proc/<PID>/lock	Si le processus est bloqué.
/proc/<PID>/nmigs	Nombre de migrations.
/proc/<PID>/where	hôte du processus.
/proc/<PID>/lock	Débloquage du processus.
/proc/<PID>/goto	Migre le processus.
/proc/<PID>/migrate	Migre le processus.
/proc/hpc/remote/from	Hôte initial du processus.
/proc/hpc/remote/identity	Informations supplémentaires.
/proc/hpc/remote/statm	Statistiques mémoire.
/proc/hpc/remote/stats	Statistiques CPU.

3.3.2 /mfs

oMFS est accessible une fois monté, typiquement en /mfs. Comme expliqué précédemment, il s'agit d'un point central d'accès aux systèmes de fichiers du cluster, et ce en fonction du nœud.

Répertoir virtuel	Répertoire associé	Description
/mfs/here	/	Système de fichiers sur lequel le processus s'exécute.
/mfs/home	/	Système de fichiers local.
/mfs/<ID>	/	Système de fichiers du nœud <ID>

3.3.3 libmos

```
#include <libmosix.h>
```

libmos est une bibliothèque C de fonctions très simples à utiliser. Nous de décrirons pas l'interface de cette bibliothèque car cela est un peu trop technique. Par exemple, nous avons nous même réalisés des petits utilitaires en C afin d'obtenir une table associant chaque nœud, désigné par un numéro, à son adresse IPv4.

3.4 Configuration

3.4.1 Le réseau

Pour des performances optimales, les différentes machines doivent être reliées par une connexion fiable haut-débit (au moins de 100Mbits/sec). L'utilisation de hubs est à proscrire du fait du trafic réseau inutile que génère ce genre de matériel d'inter-connexion passif.

Si les différentes machines du cluster sont inter-connectées par un nombre important de switchs ou de routeurs, c'est-à-dire si la topologie du réseau est relativement complexe, il est possible d'adapter la configuration du noyau lors de la compilation.

3.4.2 Le noyau

Voici les differentes options configurables dans le noyau directement :

openMosix process migration support Cette option est évidement à sélectionner, sans quoi le système sera incapable de faire migrer automatiquement les différents processus.

Support clusters with a complex network topology

Maximum network-topology complexity to support sont les options qui permettent de régler la complexité de la topologie du réseau (comme indiqué ci-dessus).

Stricter security on openMosix ports Cette option permet au cluster de détecter automatiquement la connexion d'un nouveau nœud, sans avoir à le rajouter dans la base d'adresses IP des nœuds du cluster (ce qui peut devenir fastidieux si le cluster contient un nombre important de machines). Cependant cela rend le cluster vulnérable à de possibles intrusions.

Level of process-identity disclosure Grâce à ce réglage, on peut déterminer quelles sont les informations que les autres nœuds auront la possibilité de consulter sur l'exécution d'un processus sur un nœud distant. Selon la valeur :

0 Les nœuds distants n'auront accès à aucune information.

1 Seuls le PID du processus sera consultable.

2 Toutes les machines peuvent consulter le PID, UID, GID.

3 Le PID, UID, GID, PGRP, SESSION, COMMAND, c'est-à-dire toutes les informations sont disponibles.

openMosix File-System Cette option active le support oMFS.

Poll/Select exceptions on pipes Il s'agit d'une option pour le support d'exécution de programme. En effet en utilisant `ioctl(pipefd, TCSBRK, arg)`, si `(arg & 1)` est vrai une exception est lancée et prévient qu'un processus tente de lire sur le pipe et si `(arg & 2)` est vrai cela signifie qu'il n'y a plus aucun lecteur sur le pipe.

Disable OOM Killer Permet de désactiver le *Out Of Memory Killer*. L'OOMK est un mécanisme de sécurité de Linux : lorsque le système n'a plus de mémoire libre, l'OOMK sélectionne un processus et le tue afin de rétablir une situation acceptable. Cependant, l'heuristique utilisée par l'OOMK se prête des fois mal aux gros processus qui migrent sur le cluster. De plus, l'OOMK peut entrer en action alors qu'il y a de la mémoire disponible sur un autre nœud, auquel cas il ne faut pas tuer le processus mais le faire migrer vers hôte moins chargé.

3.4.3 Le daemon

OpenMOSIX utilise un système de daemon afin de contrôler l'activité des machines présentes sur le cluster. Si l'on a activé l'option *Stricter security on openMosix ports*, il est obligatoire de renseigner sur tous les nœuds le fichier `/etc/openmosix.map` en y mettant les adresses IPv4 ou bien la plage d'adresses IPv4 des machines concernées.

3.5 Outils openMOSIX

openMOSIX est accompagné d'une suite logicielle facilitant son administration. Ces logiciels sont également disponibles sous forme de paquetage. Tous ces outils sont très faciles d'utilisation et sont accompagnées de documentation sous forme de `man`.

3.5.1 Les outils en ligne de commande

En installant openMOSIX, des outils d'administration en ligne de commandes sont fournis. Ces outils utilisent l'API précédemment décrite : leurs options de lancement sont donc en relation directe avec `/proc/hpc/admin` . Voici la liste des principaux utilitaires :

- `migrate` ordonne a un processus de migrer. La migration peut être demandée en fonction du nœud ou de la charge. On peut également ordonner au processus de retourner sur son nœud originel.

- `mosctl`
  fournit de nombreux réglages et d'informations à l'administrateur. Cet utilitaire permet de controler la vitesse d'un nœud, et d'obtenir des informations sur d'autre nœud comme la vitesse, la mémoire vive, etc.
- `moslimit`
  est un outil d'administration qui permet de connaître la charge du nœud local ou d'un nœud distant. La charge maximale, l'utilisation CPU sont aussi paramétrables avec cet outils.
- `mosrun`
  mosrun permet de lancer une commande avec des réglages différents de ceux par défaut. Si la tâche doit rester sur un nœud précis ou bien si l'on souhaite empêcher une tâche de migrer (alors que par défaut tous les processus ont le droit de migrer), il faudra donc lancer le travail avec mosrun pour un paramétrage plus fin. mosrun est enfait la partie avant d'une dizaine d'utilitaires qui permettent des réglages très spécifiques que nous ne détaillerons pas.

3.5.2 La suite openMOSIXview

La suite openMOSIXview rassemble les principales fonctions utiles pour un administrateur d'un cluster. Il s'agit d'un outil intégré graphique sous X/Windows.

Certaines opérations telles que la configuration d'un nœud nécessite un accès ssh, dans ce cas, il faudra que les clusters soit dans une zone de confiance, afin de pouvoir établir les connexions sans avoir à demander systématiquement à l'administrateur de s'identifier (tâche très fastidieuse).

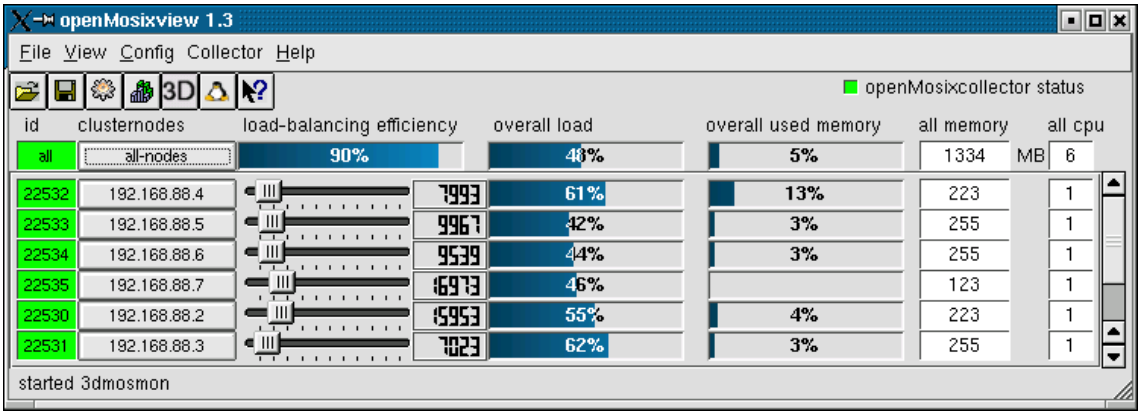

FIG. 3.1 – openMOSIXview.

La fenêtre principale est la plus utilisée car elle permet de voir immédiatement quelle est la machine la plus chargée et quelles sont celles qui ne

répondent plus.

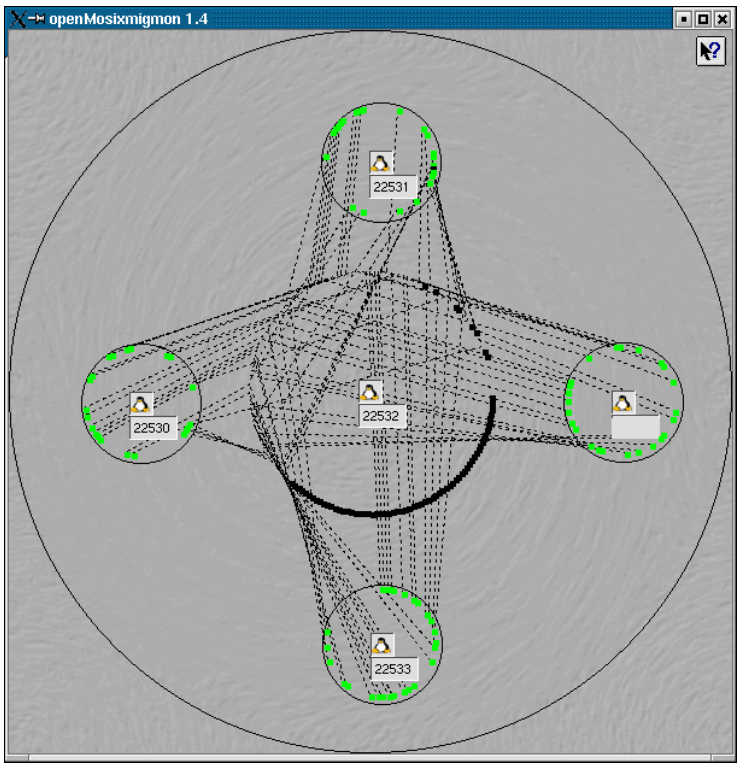

FIG. 3.2 – openMosixmigmon.

L'utilitaire openmoixmigmon qui permet de voir quel processus migre et vers quel nœud. Cette utilitaire est trés intéressant car il permet d'aprécié visuelement la migration mais il ne permet d'observer que la migration des processus sur la machine où il est executé. Sur un parc de dizaine de machines, il faudrait donc lancer autant de moniteurs que de machines.

openMosixprocs est une sorte de `top` permettant de mieux monitorer l'acitvité de chaque nœud : grâce à cet outils on est capable de voir sur quelle machine est exécuté un processus donné. Cet utilitaire nous montre aussi quels processus ne pourront jamais migrer.

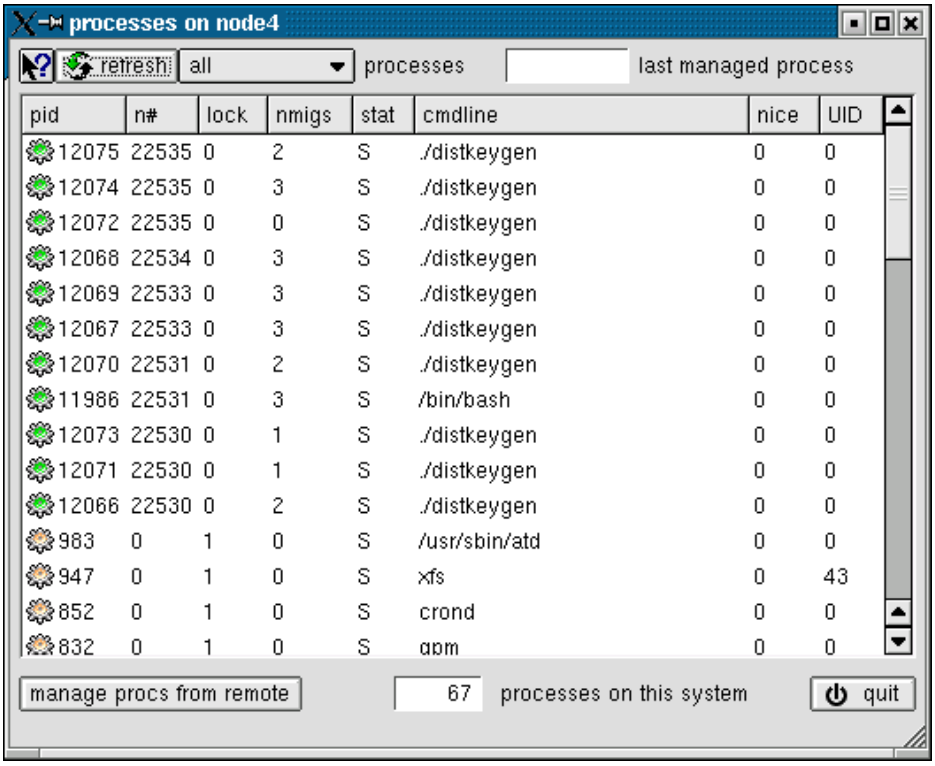

FIG. 3.3 – openMOSIXprocs.

Chapitre 4

Expérimentations

Toutes les expérimentations ont été efectuées 3 fois afin d'écarter toute aberration.

4.1 Configurations

Université-9

Ce cluster est le regroupement de 9 machines du département informatique de l'Université de Pau et des Pays de l'Adour. Chaque machine a la configuration suivante :

Processeurs	RAM
Pentium4 2 Ghz	512 Mo

Les machines sont reliées par un réseau 100MBits/s. Les expérimentations ont été effectuées à des heures où le trafic réseau est peu important : on considère donc que le bruit n'a pas d'incidence sur les résultats.

Ce cluster fonctionne sur Live-CD Knoppix [15], c'est-à-dire que le système GNU/Linux support du cluster n'est pas installé sur les machines : il est exécuté directement à partir du CD-ROM. La machine s'amorce sur le CD et le système GNU/Linux se charge. Nous avons utilisée des Quantian [13] qui est une distribution dérivée de ClusterKnoppix [14], c'est-à-dire une distribution Knoppix avec un noyau openMOSIX ainsi que les outils nécessaires à la surveillance et l'administration du cluster.

Maison-5

Nous avons pu rassembler chez nous 5 machines :

Processeurs	RAM
Pentium Celeron 830Mhz	512 Mo
Athlon XP 2000+ 1666Mhz	1024 Mo
Pentium Celeron 700Mhz	128 Mo
Athlon 1000Mhz	128 Mo
Bi-Pentium Xeon 2400Mhz	2048 Mo

Toutes ces machines tournent sous le système GNU/Linux Debian avec comme noyau un 2.4.22 patché avec openMOSIX et sont reliées par un réseau local 100Mbits/sec passant par un switch.

Nous avons choisit le système Debian car c'est celui que nous utilisons tous les jours sur nos machines personnelles. Pour l'anecdote, nous avons abandonné l'installation des RPM openMOSIX pour RedHat tellement cela est problématique (les openMOSIXtools ont refusés de fonctionner à cause d'une erreur interne au noyau) et consommateur de temps alors que nous avons manuellement patché les noyaux de nos machines sous Debian en un instant pour obtenir un cluster fonctionnel en quelques dizaines de minutes. Nous recommandons cette méthode, car une configuration fine et la compilation manuelle du noyau donne de très bons résultats par rapport aux noyaux génériques et peu optimisés fournis avec les distributions classiques.

4.2 Traitements parallèles sur un même fichier : pipeline

4.2.1 Protocole

Ce banc d'essai a été réalisé sur le cluster Université-9. Le test consiste à appliquer une série de traitements successifs à un fichier. Ces différents traitements permettent de paralléliser le procédé : un traitement N+1 n'a pas besoin de la complétion du traitement N pour pouvoir débuter.

Taille du fichier : 76Mo (tarball OpenOffice.org 1.1)

Traitements : le banc d'essai se présente sous la forme d'un simple script. Ce script calcule l'empreinte MD5 du fichier, puis le compresse au format bzip2, code en base64, le décode, le décompresse et en recalcule l'empreinte MD5 (cette dernière étape permettant de vérifier l'intégrité du fichier).

```
   md5sum $1 &
cat $1 | bzip2 | uuencode - | uudecode | bunzip2 | md5sum
```

4.2.2 Observations

Le résultat sur le cluster est légèrement supérieur. On le considère comme équivalent au résultat sur une seule machine.

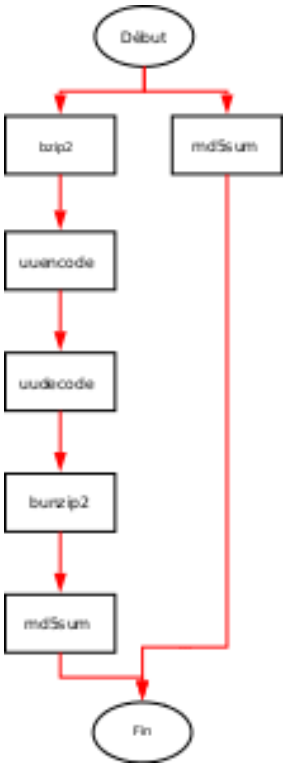

FIG. 4.1 – Pipeline

openMOSIXview ne rend compte d'aucune migration de processus.

Nombre de nœuds	Temps
1	2 minutes 16s
9	2 minutes 18s

4.2.3 Interprétation

Comme attendu, les résultats d'utilisation sur le cluster sont inefficaces. On n'a aucune accélération car bien que les processus soient parallèles, ils sont fortement limités en entrées/sorties. Le coût des communications inter-processus est bien trop élevé pour qu'on puisse tirer avantage du cluster. openMOSIX prend donc à raison la bonne décision : ne pas faire migrer les processus. Cela peut paraître décevant, mais c'est avant tout une décision rationnelle limitant le trafic réseau, facteur limitant dans les clusters. Ce genre de traitement ne se prête apparemment pas bien à la migration.

4.3 Rendu d'image

4.3.1 Protocole

Le cluster Maison-5 a été utilisé pour réaliser ce test. Le but de ce test est de faire un rendu d'image de synthèses grâce aux logiciel yafray[17] qui permet de choisir le nombre de processeurs, donc de nœuds, qu'il doit utiliser. Une version de YafRay patchée a été nécéssaire afin de pouvoir utiliser le cluster, openMOSIX pourra décider de faire migrer ou non les différents processus du rendu.

4.3.2 Observations

Les différents processus migrent bien et les nœuds plus rapides devenus oisifs reprennent les travaux des machines moins puissantes. Les processus ne font donc pas de va-et-viens ce qui aurait tendance à surcharger inutilement le réseau et être contre-performant.

Nb nœuds	Qualité	Temps Rendu	Image n°
1	low	1h00	1
1	best	15h07	2
5	low	0h22	1
5	best	4h21	2

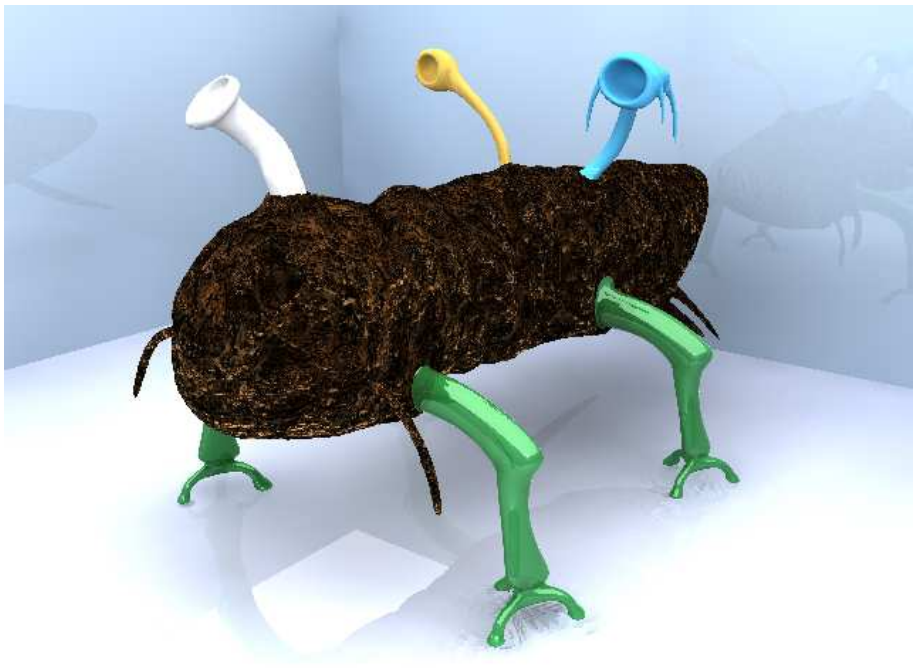

FIG. 4.2 – Rendu n°1

FIG. 4.3 – Rendu n°2

4.3.3 Interprétation

La séparation des différentes tâches lors d'un rendu est tout à fait exploitable par un cluster à répartition de charge tel qu'openMOSIX. En effet le gain de performance n'est pas négligeable, sur un rendu d'une image de dimensions 640x480 pixels, le temps de calcul passe de 15h07 à 4h21 ce qui donne un gain d'environ 3,5.

Les communications inter-processus étant minimes, le système décide à juste titre de faire migrer les tâches vers les machines les moins chargées. Le système est même capable de ré-attribuer un processus, qui a déjà migré, à une autre machine plus puissante ou moins chargée : la sélection des machines se fait donc bien suivant la charge (cluster à répartition de charge) et la puissance disponible.

La version actuelle d'openMOSIX est incapable de faire migrer des processus utilisant de la mémoire partagée. Mais il existe un patch pour YafRay qui utilise les processus plutôt que les threads. Nous avons donc appliquer ce patch. Certains utilisateurs de YafRay font part de leur expérience avec un cluster de 19 machines et ont constaté une accélération proche de 19, c'est-à-dire une efficacité égale à 1.

Ce genre d'application exploite vraiment les possibilités d'un tel cluster et peut justifier un investissement pour la création d'un cluster plutôt que l'achat d'un seul super-calculateur.

4.4 Compression de pistes audio

4.4.1 Protocole

Il s'agit de compresser les pistes d'un CD-Audio. Les machines actuelles s'acquittent de cette tâche rapidement mais cette expérimentation est représentative d'un certain type de travail : l'application d'un même procédé à un ensemble de données, toutes les tâches étant indépendantes. Nous y reviendrons plus tard.

Après avoir extrait les 12 pistes d'un CD-Audio Punk/Rock au format Wave, il s'agit de compresser chacune des ces pistes au format OGG avec l'algorithme Vorbis, qualité 10. Il s'agit d'une compression destructive et très efficace.

Voici les 2 scripts utilisés pour la compression :

Compression séquentielle

```
for f in *.wav;
 do
   out=$(basename $f).ogg;
   oggenc -q 10 $f -o $out;
```

```
done
```

Compression parallèle

Notez bien l'& final qui a pour effet de détacher la tâche et de passer à l'instruction suivante. Le `wait` final impose d'attendre que toutes les tâches soient terminées.

```
for f in *.wav;
 do
   out=$(basename $f).ogg;
   oggenc -q 10 $f -o $out &
 done

wait
```

4.4.2 Observations

openMOSIXview montre très bien le phénomène de migration. Les processus sont répartis de manière équitable sur les différents nœuds.

Nombre de nœuds	Temps séquentiel	Temps parallèle
1	5 minutes 9s	5 minutes 9s
9	5 minutes 9s	1 minutes 7s

On constate que la compression d'une piste nécessite 26 secondes (en moyenne) de temps CPU.

4.4.3 Interprétation

openMOSIX fonctionne à merveille ! Le temps nécessaire pour compresser toutes les pistes a été divisé par 4,5. Le résultat est tout à fait prévisible : 12 tâches sont à exécuter. Sur le cluster de 9 machines, 9 tâches peuvent s'exécuter en parallèle ; un fois les 9 premières pistes compressées, il reste 4 pistes qui sont attribuées à 4 machines. Le temps final de 67s concorde avec cette analyse. Cet exemple montre une très bonne accélération et une efficacité modérée : ceci est du au petit nombre de données. Dans le cadre de la compression de plusieurs centaines de pistes, l'efficacité serait alors maximale.

openMOSIX est donc très performant dans la répartition de processus indépendants et gourmands en CPU.

Motivés par ces résultats, nous avons tentés de compiler la suite GCC mais nous avons très vite arrêtés les tests : la compilation d'une unité de traduction ($\approx$ fichier) est une tâche trop courte (en moyenne 4 secondes de temps CPU). Le surcoût à la migration est donc sensible et openMOSIX de fait pas migrer ces tâches trop courtes.

4.4.4 Extrapolation

openMOSIX est doué pour répartir les travaux pouvant être subdivisés en plusieurs processus indépendants. La compression de pistes audio en est exemple. openMOSIX se serait montré aussi efficace s'il avait s'agit de compresser les chapitres d'un film DVD ou de réaliser des tâches industrielles : traitement de nombreux fichiers ou d'images.

Nous n'avons pas été en mesure de trouver au sein même du département des travaux de ce type et qui se prêtent parfaitement au clustering à répartition de charge.

4.5 Utilisation normale

Nous avons procédés à des expérimentations informelles d'openMOSIX dans le cadre d'une utilisation normale : navigation sur Internet, traitement de texte, compilation de petits programmes, visualisation d'images.

Ces tâches ne sont pas suffisamment intensives où ne sont pas éligibles (mémoire partagée dans le cas d'OpenOffice.org). Nous avons constaté aucune migration, donc aucune accélération ni ralentissement.

Les seules migrations que nous avons observées ont un caractère occasionnel : il s'agissait de processus fous, c'est-à-dire exécutant une boucle infinie et consommant beaucoup de temps CPU. Il nous fallut un certain temps pour réaliser que ces processus avaient migrer : en effet, sur une machine seule, un ralentissement notable indique la défaillance d'un processus. Il n'en est rien avec openMOSIX, le processus fou ayant migré sur un nœud distant.

4.6 Conclusions des bancs d'essai

Le bilan de cette série de tests est très contrasté et sans équivoque : en ce qui concerne openMOSIX, c'est tout ou rien. Les résultats établissent clairement que dans le cas d'applications qui ne sont pas orientées cluster, la répartition de charge ne fonctionne pas. Au contraire, dans le cas d'applications pouvant être divisées en sous-tâches indépendantes, l'accélération est maximale, l'efficacité optimale. De plus, la répartition de charge fonctionne efficacement et équitablement.

4.7 Aspect financier

Les bancs d'essais concluant sont particulièrement intéressant : en effet, avec peu de moyens (une ferme de PC domestiques) nous avons pu obtenir une grande puissance de calcul. L'agrégation de 9 PC de l'université a également formée un ordinateur puissant. Une comparaison avec des supercalculateurs s'impose : openMOSIX est effectivement moins souple qu'un

calculateur multi-processeur, mais sous réserve de quelques contraintes, son fonctionnement macroscopique est le même (SSI). Il en faut pas non-plus oublier les solutions à la MPI qui donnent des résultats excellents. Tous les processeurs du cluster sont utilisés comme ce serait le cas sur un système SMP. L'aspect financier est évidemment très important à ce niveau de comparaison ; voici quelques chiffres. Nous avons effectué des comparaison avec des machines équivalentes en terme de puissance de calcul, de capacité mémoire et de stockage.

Machine	Prix
Maison-5	2 200 €
IBM xSeries 255 - 4 Processeurs	$14 972
Université-9	6 000 €
IBM xSeries 445 - 8 Processeurs	$65 799

Deux constations : les calculateurs sont excessivement plus chers que les solutions de clustering ; leur prix croit de manière exponentielle (linéaire pour les clusters).

Un autre point est à prendre en compte : la mise-à-jour du matériel. Avec un super-calculateur, il s'agit d'acheter une nouvelle machine. Avec un cluster, l'augmentation de la puissance de calcul s'obtient en rajoutant des nœuds, ce qui est encore une fois bien moins onéreux. Le contrôle du nombre de nœuds permet aussi d'obtenir exactement la puissance que l'on désire (ou pour laquelle on souhaite investir).

Il faut également considérer le cas d'une entreprise ne nécessitant un grande puissance de calcul que sporadiquement : elle opte généralement pour l'achat d'un calculateur alors sous-utilisé ou bien elle achète du temps de calcul dans un centre. Avec openMOSIX, la puissance inutilisée de son parc informatique peut remplir cette tâche pour un coût quasiment nul.

En somme, un supercalculateur est un investissement extrêment lourd. Le clustering et openMOSIX fournissent une solution adaptée, très bon marché et tout aussi performante.

Chapitre 5

Conclusion

Il ressort de cette étude détaillée sur le clustering et openMOSIX plusieurs points importants à la compréhension du clustering et de ses applications.

Le clustering a été inventé pour répondre à différents besoins mais toujours dans le but de proposer des solutions beaucoup moins onéreuses que les calculateurs des grands fabriquants.

openMOSIX est un cluster à hautes performances et à répartition de charges ; sur ces point il est irréprochable : il délivre une puissance brute équivalente à celle d'un super-calculateur. Cette puissance n'est apprivoisable qu'à certaines conditions, principalement applicatives : les tâches doivent se subdiviser en processus indépendants.

openMOSIX a permit de réaliser tout au long de nos expérimentations des calculs longs et fastidieux en des temps très réduits. Nous avons était surpris de pouvoir bénéficier d'une telle accélération sur un notre modeste cluster domestique, donc pour un coût dérisoire. Ceci n'est qu'un aperçu des possibilités d'openMOSIX en milieu industriel.

openMOSIX est une solution complète : elle fournit des outils variés de configuration, administration et surveillance, une API riche et une large documentation. Notre étude montre que la réalisation d'un cluster est chose aisée et nécessite un niveau peut élevé de connaissances pour son exploitation.

Il apparaît clairement que le clustering, et l'une de ses meilleures solutions – openMOSIX –, proposent une alternative de qualité aux super-calculateurs. Aujourd'hui, on compte 6 clusters parmi les 10 ordinateurs les plus puissants de la planète. La première position est encore occupée par un super-calculateur, mais gageons qu'il sera vite détrôné. En effet, les super-calculateurs montrent leurs limites au niveau matériel (limite physique du silicium), leurs puissances plafonnent tandis que leurs coûts explosent. Les clusters ont une évolution inverse : leurs puissances ne cessent d'augmenter (il suffit de rajouter des machines, elles aussi de plus en plus puissantes) alors

que le prix unitaire d'un micro-ordinateur n'a jamais été aussi bas.

Le clustering et openMOSIX sont donc une solution d'avenir parfaitement adaptée à des tâches de toutes envergures.

Bibliographie

[1] GNU's Not Unix
http ://www.gnu.org/

[2] Licence Publique Générale GNU
http ://www.gnu.org/licenses/gpl.html

[3] Licence de Documentation Libre GNU
http ://www.gnu.org/copyleft/fdl.html

[4] Pixar Animation Studios
http ://www.pixar.com

[5] « Le Monde de Nemo »
http ://www.disney.fr/FilmsDisney/nemo

[6] LightWave Rendering
Modeler et moteur de rendu 3D
seul le moteur de rendu fonctionne sous GNU/Linux.
http ://www.newtek.com/news/releases/07-23-02-c.html

[7] Alias Wavefront Maya
Logiciel d'animation et d'effet 3D.
http ://www.alias.com/

[8] Noyau Mach
Université de Carnegie-Mellon, États-Unis
http ://www-2.cs.cmu.edu/afs/cs/project/mach/public/www/mach.html

[9] Distributed Compiler
http ://distcc.samba.org/

[10] Algorithme Lempel-Ziv-Oberhumer (LZO)
http ://www.oberhumer.com/opensource/lzo/

[11] Network File System
http ://nfs.sourceforge.net/.

[12] Parallel Virtual File System
http ://www.parl.clemson.edu/pvfs

[13] Quantian
Distribution ClusterKnoppix orientée scientifique *http ://-dirk.eddelbuettel.com/quantian.html*

[14] ClusterKnoppix
Distribution Knoppix modifiée utilisant un noyau openMOSIX.
http ://bofh.be/clusterknoppix/

[15] Knoppix
Knoppix est un CD-ROM auto-amorçable avec un système d'exploitation(GNU/Linux pour ordinateurs de type PC, une collection de logiciels, la détection automatique du matériel.
http ://www.knoppix.org/

[16] User Mode Linux
http ://user-mode-linux.sourceforge.net/

[17] YafRay
http ://yafray.org

[18] RFC XXXX

[19] RFC XXXX

[20] SourceForge
Répertoire/hébergeur de projets et applications libres.
http ://sourceforge.net/

[21] Message Passing Interface
http ://www.mpi-forum.org/

[22] Parallel Virtual Machine
http ://www.csm.ornl.gov/pvm/