REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE

ECOLE NATIONALE SUPERIEURE POLYTECHNIQUE

Departement d'Electronique

Projet de Fin D'Etudes

En vue de l'obtention du Diplome
d'Ingénieur d'Etat en Electronique

Theme

Mise en œuvre d'un cluster sous linux : Application au calcul parallèle

Proposé et dirigé par :

- Mr R. SADOUN

Réalisé par :

- Mr. LAIB Rabie
- Mr. TALEB Abdelkrim

Promotion : Juin 2009

Remerciment

Avant d'entamer notre présent rapport, nous tenons à adresser nos remerciements à l'ensemble des enseignants qui nous ont assistés pour que ce projet de fin d'études soit fructueux et profitable.

Ensuite, toutes nos pensées de gratitude se dirigent vers Monsieur R. Sadoun pour bien avoir voulu encadrer notre projet, pour son aide et les renseignements précieux qu'il nous a fourni. On remercie très chaleureusement les membres de jury pour l'honneur qu'ils nous ont fait en acceptant d'être les rapporteurs de ce mémoire.

Nous voudrions également remercier toutes les personnes qui par leur aide ou simplement par leur gentillesse ont contribué d'une façon ou d'une autre à la bonne réalisation du projet dont nous avons eu la charge. Nous pensons particulièrement à l'équipe du centre de calcul.

Table des matières

Table des figures

Introduction

De nos jours et dans le domaine d'ingénierie, la nécessité d'augmenter les capacités de calcul ou d'assurer la disponibilité d'un service donné, se fait de plus en plus sentir. Le recours aux solutions de parallélisme peut répondre à ce besoin, ces solutions peuvent être des architectures s'appuyant sur des algorithmes exhibant le parallélisme.

Le recours a été perçu en réponse aux limitations de l'exécution séquentielle. Les limites de cette dernière découlent des limites technologiques des microprocesseurs généralistes : performances intrinsèques, temps d'accès mémoire.

A l'image de ce développement de ces solutions, les supercalculateurs, (par exemple CRAY X1, BlueGene/P , Juropa...)[1] se présentent comme des dispositions à ce besoin. Leurs coûts et leur accès s'avère une contrainte majeure même si les performances restent plus qu'intéressante. Ce fut un frein à la démocratisation du calcul parallèle.

Le clustering est une solution alternative, avec un rapport prix/performance qui satisfait souvent les besoins demandés. Dans ce qui suit,on va mettre en avant quelques solutions de clustering , avec différents types de configurations dans le but de la performance, Le choix a été porté sur deux solutions qui utilisent deux principes différentes. La première est OpenMosix qui est un patch pour noyau linux ; la seconde est PVM , qui est une librairie pour C ou Fortran.

Chapitre 1

Parallélisme et architectures parallèles

1.1 Introduction

L'architecture des ordinateurs, qu'il s'agisse de microprocesseurs ou de super-calculateurs, est fortement influencée par l'exploitation d'une propriété fondamentale des applications : le parallélisme. Un grand nombre d'architectures présentes sont parallèles. Ce type d'architecture touche une large gamme de machines depuis les PC biprocesseurs jusqu'aux supercalculateurs. Aujourd'hui, la plupart des serveurs sont des machines parallèles à gros grains.[2]

1.2 Définition du parallélisme

Le parallélisme est un nom masculin (grec parallêlismos, du grec classique parallêlos) qui est une technique d'accroissement des performances d'un système informatique fondée sur l'utilisation simultanée de plusieurs processeurs. (Quand le nombre de processeurs est très important, on parle de parallélisme massif).[3]

Bernstein a introduit en 1966 un ensemble de conditions permettant d'établir la possibilité d'exécuter plusieurs programmes (processus) en parallèle. Supposons deux programmes : P1 et P2. Supposons que chacun utilise des variables en entrée et produise des résultats en sortie. Nous parlerons des variables d'entrée de P1 et P2 (respectivement E1 et E2) et des variables de sortie de P1 et P2 (respectivement S1 et S2).

Selon Bernstein, les programmes P1 et P2 sont exécutables en parallèle (notation : $P1 \| P2$) si et seulement si les conditions suivantes sont respectées : $E1 \cap S2 = 0, \quad E2 \cap S1 = 0, \quad S2 \cap S1 = 0$

Plus généralement, un ensemble de programmes P1, P2... Pk peuvent être exécutés en parallèle si et seulement si les conditions de Bernstein sont satisfaites, c'est-à-dire si $Pi \| Pj$ pour tout couple (i, j) avec $i \neq j$. [2]

1.3 Types de parallélisme

Il existe deux types de parallélisme :

- Parallélisme de données.
- Parallélisme de contrôle.

1.3.1 Parallélisme de données

C'est l'exécution simultanée d'une même opération par un ou plusieurs processeurs sur différentes données. [2]

Exemple : prenons un exemple simple d'algèbre linéaire et plus particulièrement de calcul matriciel. L'addition de deux matrices consiste, pour tous les éléments de mêmes indices des deux matrices opérandes à les additionner et à stocker le résultat dans l'élément de même indice de la matrice résultat. Voici la boucle correspondante pour les matrices opérandes B et C et la matrice résultat A :

```
Pour i de 1 à n

   Pour j de 1 à n

      A [ i ][ j ] <- B [ i ][ j ] + C [ i ][ j ]

   FinPour

FinPour
```

Comme pour la boucle étudiée précédemment, les itérations de cette boucle sont indépendantes. Il y a n^2 itérations avec une opération par itération. Le potentiel de parallélisme exploitable dans cette boucle est donc de n^2 opérations simultanées.[2]

1.3.2 Parallélisme de contrôle

C'est un type de parallélisme pour lequel des opérations différentes sont réalisées simultanément. Ce parallélisme peut provenir de l'existence dans le programme de fonctions indépendantes. Il peut aussi provenir d'opérations indépendantes dans une suite d'opérations. Ce parallélisme ne dépend donc pas des données mais de la structure du programme à exécuter. C'est l'absence de dépendances entre différentes parties du programme (quelle que soit leur taille : fonctions, boucles, opérations) qui est la source du parallélisme de contrôle.

Exemple : voici un exemple très simple de programme pour un serveur :

```
faire toujours
   détecter (demande_client)
      si (demande client = vrai)
         lancer (traitement_client)
      finsi
finfaire
```

Voici le programme correspondant pour le traitement client :

```
début
   ...
   ouvrir (fichier_client)
   faire (traitement_demandé)
   fermer(fichier_client)
   ...
fin
```

Supposons que la fonction lancer () retourne immédiatement après avoir lancé le programme traitement_client ; c'est-à-dire sans attendre que celui-ci se termine. Dans ce cas, le programme du serveur continue à s'exécuter alors que le programme traitement_client est en cours d'exécution. S'il y a suffisamment de ressources, ces deux programmes seront exécutés simultanément. Si le serveur reçoit une nouvelle demande_client, il lancera l'exécution du nouveau traitement_client de la même manière. Il est donc possible, à un instant donné, que plusieurs traitement_client s'exécutent en même temps (avec des niveaux d'avancement différents) en plus du programme serveur. Comme ces programmes peuvent réaliser des opérations différentes, il ne s'agit pas de parallélisme de données mais bien de parallélisme de contrôle.[2]

Un autre exemple est illustré dans la figure 1.1

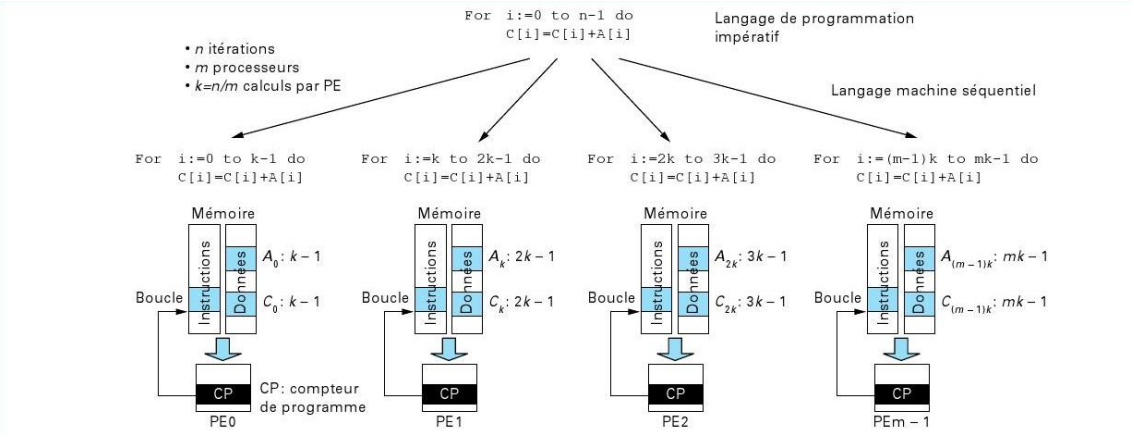


FIG. 1.1 – Principe d'exécution d'une boucle parellèle avec le modèle multiséquentiel

1.4 Classification des architectures parallèles

Pour distinguer des familles d'architectures, on a recoure aux classifications suivantes :

1.4.1 Classification de Flynn

Elle est basée sur la multiplicité du flot d'instructions et le flot de données :
- SISD : Single Instruction Single Data Stream (séquentiel)
- SIMD : Single Instruction Multiple Data Stream
- MISD : Multiple Instruction Single Data Stream (vectoriel)
- MIMD : Multiple Instruction Multiple Data Stream

1.4.2 Classification suivant le grain de calcul

La recherche de la performance a conduit à explorer un vaster espace de configuration possibles pour les machines parallèles.

Un des axes de cet espace est celui du grain de calcul. Pour une même performance globale, il existe de nombreux doublets possibles (nombre de processeurs, performance par processeur). Les machines SIMD comme la Connection machine 2 de Thinking Machine Corp. et les multiprocesseurs Vectoriels (aussi appelés PVP) représentent les deux extrémités de cet axe. La Connection Machine 2 pouvait comporter jusqu'à 65 536 processeurs 1 bit. Dans cette machine, chaque unité de traitement réalisait seulement des opérations sur des opérandes 1 bit. Dans ce cas, le grain de calcul est dit fin. Un très grand nombre de processeurs est donc nécessaire pour atteindre des performances importantes. Les multiprocesseurs Vectoriels regroupent peu (de l'ordre de 32) de processeurs très performants fonctionnant à une fréquence très élevée et calculant directement sur

64 bits. Ce sont des architectures à gros grain de calcul. Les calculateurs parallèles à base de microprocesseurs standards possèdent un grain moyen. Les microprocesseurs atteignent des performances importantes se rapprochant des performances des processeurs vectoriels. Il n'est donc pas nécessaire d'en assembler beaucoup pour atteindre des performances élevées.[2]

1.5 Systèmes répartis

C'est un ensemble de machines (ordinateurs, processeurs) interconnectées. Chaque machine est autonome et exécute des processus et communique avec les autres. Tous ces processus coordonnent leurs activités de tel sort que tous le système paraît comme une seul machine capable d'accomplir un traitement unique. (Voir illustration figure 1.2

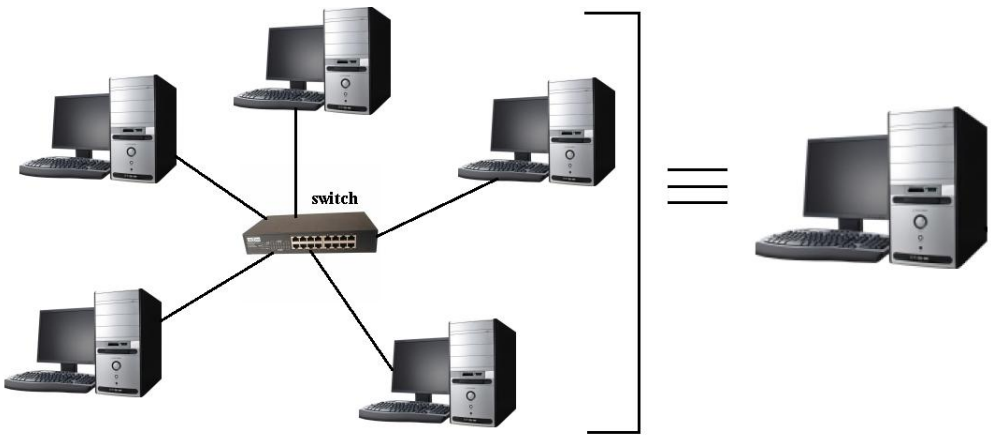


FIG. 1.2 – Illustration d'un système réparti

1.6 Conclusion

Les architectures parallèles sont nées de la conjonction de trois éléments :
- les besoins des applications,
- les limites des architectures séquentielles,
- et l'existence dans les applications de la propriété de parallélisme.

Il existe plusieurs options dans les architectures parallèles, chacune d'elles offre des avantages bien spécifiques. Dans ce qui va suivre on se propose de focaliser notre étude sur les clusters comme option d'architecture parallèles.

Chapitre 2

Le Clustering

2.1 Introduction

Les premieres publications sur le clustering ont apparues dans les années 60. On parlait alors principalement de partage des ressources sur un réseau. C'est au début des années 90, avec les développements des interfaces réseau à haut-débit et au développement des langages et des concepts, que le clustering a pris toute son importance.

Aujourd'hui, le marché des clusters représente 3% ou 4% des ventes des serveurs. C'est un marché de niche qui a cependant subit un fort taux de croissance ces dernières années. En effet, les entreprises font de plus en plus appel à ce marché, séduites par leur excellent rapport prix/performance. Ce type de technologie ouvre de nombreuses perspectives que ce soit dans le domaine des sciences ou sur le plan purement industriel ou commercial.[4]

2.2 Définition

Un cluster peut être défini comme un ensemble de systèmes interconnectés qui partagent des ressources de façon transparente. Chacun de ces systèmes peut être considéré comme un système à part entière puisqu'il dispose de son propre ensemble de ressources telles que processeur(s), mémoires, dispositifs d'entrées-sorties... et qu'il fonctionne sous le contrôle de sa propre copie du système d'exploitation. On appelle aussi nœuds les systèmes qui composent un cluster. [2]

2.3 Spécifications

Le cluster doit répondre à un certain nombre d'exigences afin de garantir aux utilisateurs un bon fonctionnement ; qui sont :

- Le système doit pouvoir fonctionner (au moins de façon dégradée) même en cas de défaillance de certains de ses éléments.
- Le système doit pouvoir résister à des perturbations du système de communication (perte de messages, déconnexion temporaire, performances dégradées).
- Le système doit pouvoir résister à des attaques contre sa sécurité (tentatives de violation de la confidentialité et de l'intégrité).
- Le système doit pouvoir facilement s'adapter pour réagir à des changements d'environnement ou de conditions d'utilisation.
- Le système doit préserver ses performances lorsque sa taille croît (nombre d'éléments, nombre d'utilisateurs, étendue géographique).

2.4 Typologies des clusters

Les 3 types de clusters les plus communs sont les clusters à haute disponibilité (High availability), à haute performances (High performance computing) et à répartition de charge (Horizontal scaling or load-balancing). Il faut savoir que leurs propriétés peuvent être mélangées selon les besoins spécifiques de l'utilisateur. Il existe autant de structures de clustering que de problèmes.

2.4.1 Cluster à haute performances

Les clusters à haute performances (High performance computing) consistent un ensemble d'ordinateurs liés entre eux pour fournir un maximum de puissance dans la résolution d'un problème (voir figure 2.1). Le cœur de ces clusters est formé de nœuds de calcul qui recevront le code à exécuter. Sur les plus petits on en compte une dizaine tandis que les plus grands actuels en possèdent plus de 20 000. L'architecture réseau mise en place pour la communication entre les nœuds devient alors très lourde et trés coûteuse. C'est elle qui limite en général ses performances. Il faut savoir en outre que le rapport entre le nombre de nœuds et les performances de ce type de cluster n'est pas linéaire. Il faut en effet que le programme exécuté soit fortement parallélisable et qu'il ne nécessite que peu de communication entre les unités de calcul.

2.4.2 Cluster à haute disponibilité

Les clusters haute disponibilité (High availability) (figure 2.2) sont construits pour fournir un environnement sécurisé, et tolérant face aux pannes. La méthode la plus souvent utilisée est la redondance. Elle consiste à multiplier le matériel qui pourrait être sujet à une panne. Les applications serveur étant installées de façons identiques sur les nœuds du cluster. Celle-ci existe par exemple dans les stations de travail au niveau des unités de stockage sous la forme du RAID 1 ou du RAID 5 (Redundant Array of

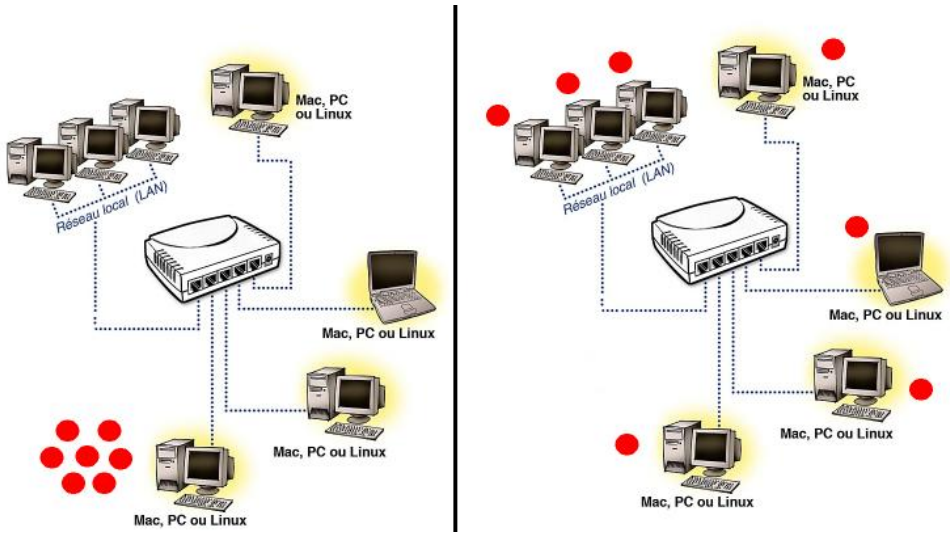


FIG. 2.1 – Illustration du clacul dans un cluster à haute performance

Inexpensive Disks). Dans le cas le plus simple on peut imaginer deux ordinateurs reliées à un disque partagé. Pendant qu'une application s'exécute sur le premier, le second est prêt à prendre le relais en cas de panne. Lorsque celle-ci survient, le second système prend le relais et s'approprie les ressources (stockage, réseau, adresses, etc.). Ce processus est appelé "failover". Il est transparent pour les utilisateurs finaux qui n'ont pas besoin de savoir sur quel ordinateur tourne leur service.

2.4.3 Cluster à répartition de charge

Les clusters à répartition de charge (figure 2.3) sont utilisés pour fournir une seule interface pour un ensemble de ressources qui peut s'accroitre de manière arbitraire. On peut imaginer un serveur web qui redirige des requêtes de clients vers un autre nœud lorsqu'il a atteint sa limite de charge. On parle alors de "load-balancing" ou de répartition de charge. Seul le nœud qui s'occupe de la répartition est visible de l'extérieur. On remarque en outre que ce type de cluster est un mélange des deux autres types. En effet, la panne d'un des nœuds reste transparente pour l'utilisateur comme ce serait le cas sur un cluster à haute disponibilité, tandis que la charge peut être répartie au mieux sur l'ensemble des nœuds dans le principe des clusters à haute performance.

2.5 Architecture d'un cluster

Quel que soit le type de tâche que devra effectuer un cluster, on peut séparer le travail à fournir en fonctions logiques. Ces fonctions logiques peuvent être groupées sur un même nœud physique ou, au contraire, peuvent être distribuées sur plusieurs

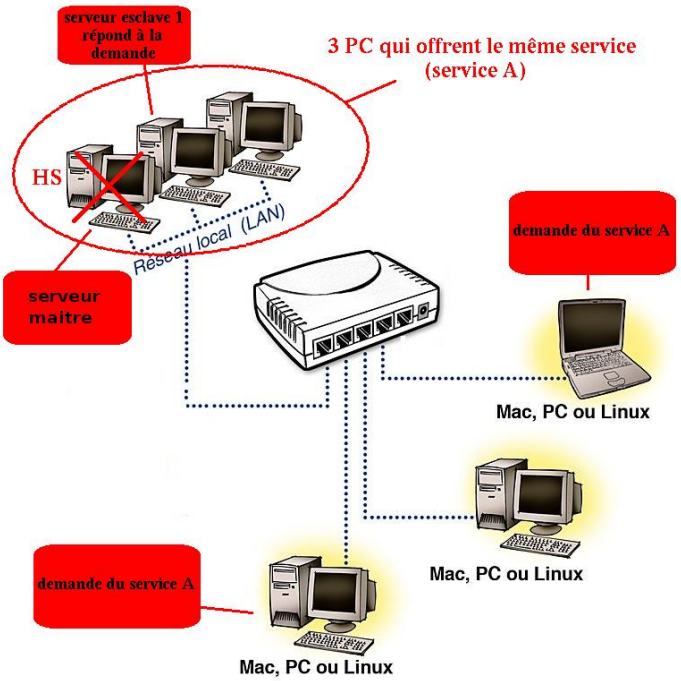


FIG. 2.2 – Illustration d'un cluster de haute disponibilité

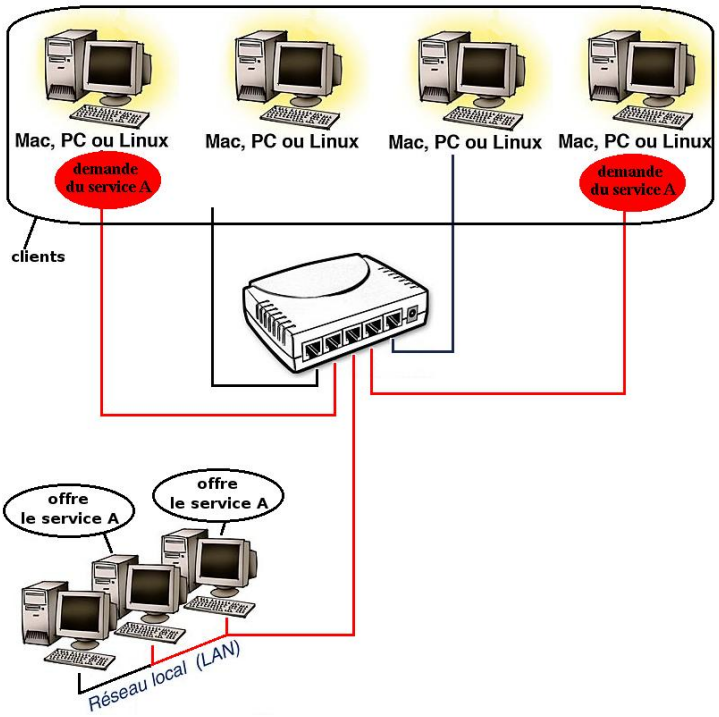


FIG. 2.3 – Illustration d'un cluster à équilibrage de charge

nœuds. Cette séparation peut très rapidement devenir complexe, mais permet d'assurer un meilleur fonctionnement et de meilleures performances. On dénombre 3 fonctions principales dont la gestion, le calcul et le stockage.

2.5.1 Nœuds de gestion

Les nœuds de gestion, communément appelés "head nodes", permettent le contrôle du cluster. Ils fournissent un ou plusieurs des services suivant :

Nœuds utilisateurs

Les nœuds utilisateurs permettent de fournir un accès (qui peut être externe) aux ressources du cluster pour proposer une tâche ou récupérer les résultats d'une tâche précédemment effectuée. Ils sont généralement sur un sous réseau qui n'est pas accessible de l'éxterieur pour des raisons de sécurité.

Nœud de management

Il permet de monitorer l'état des autres nœuds, de modifier leur configuration ou de corriger un problème. Il gère les alarmes ou les événements provenant du cluster. C'est sans doute un des nœuds dont l'importance est la plus sous-estimée. Il est capital lorsqu'on doit manager un grand nombre de machines.

Nœud d'installation

Les nœuds de calcul peuvent être régulièrement mis à jour, reconfigurés ou réinstallés. Un nœud d'installation permet de faire cette tâche de manière simple en fournissant une image unique à tous les nœuds.

Nœud de contrôle

Le nœud de contrôle fournit les services nécessaires aux autres nœuds pour assurer leur cohésion. Il existe deux groupes typiques de fonctions :

- Dynamic Host Configuration Protocol (DHCP), Domain Name System (DNS), ou d'autres fonctions similaires. Ces fonctions permettent la modularité, un nœud pouvant être ajouté ou retiré facilement tout en assurant la communication entre les nœuds.

- Scheduling, le nœud de contrôle permet de distribuer les tâches aux nœuds de calcul. De cette manière lorsqu'un nœud termine sa tâche, le nœud de contrôle peut lui en attribuer une nouvelle.

2.5.2 Nœuds de calcul

Les nœuds de calcul forment le cœur des clusters HPC. Tous les autres nœuds sont choisis et dimensionnés en fonctions de ceux-ci. C'est eux qui fourniront la trés grande partie du travail. Ils sont regroupés et chacun d'entre eux reçoit du nœud de contrôle une ou plusieurs tâches à exécuter.

2.5.3 Nœuds de stockage

Dans certains cas les unités de stockages peuvent être directement rattachées aux unités de calcul ou aux nœuds de contrôle, mais il peut devenir obligatoire d'avoir une unité de stockage indépendante. Dans ce dernier cas, le nœud de stockage gère l'accès à ce sous-système. Sa tâche étant elle aussi très particulière, le matériel qu'il nécessite peut être très spécifique pour qu'il ne soit pas lui non plus un facteur limitant les performances.

2.6 Conclusion

Le clustering est devenu aujourd'hui un élément indispensable aussi bien dans le domaine scientifique ou commerciales. Il permet d'attaquer les problèmes de fiabilité, de disponibilité, d'évolutivité et de performance, qui seraient difficile ou coûteux de résoudre autrement.

Il existe de multiples solutions de clustering caractérisées par leur aspect propriétaire ou libre.La représentation de queluque types de cluster est donnée dans la figure 2.4. Les solutions propriétaires, comme celle proposées par IBM, SUN, Hewlett Packard ...etc répondent aux besoins croissants des entreprises en termes de disponibilité et de répartition de charge. Néanmoins, elles restent :

- fermées
- généralement spécifique à un matériel donné
- onéreuses
- applications limités (peu de développement communautaire)

La réponse à ces contraintes provient des solutions libres.

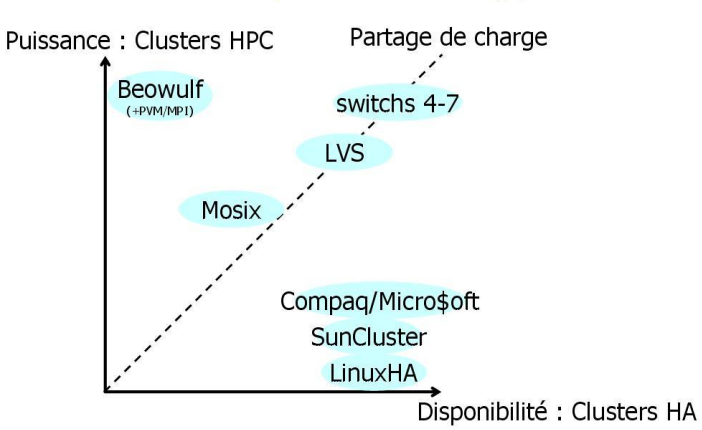


FIG. 2.4 – Différents types de clusters

Chapitre 3

Clustering Open-Source

3.1 Introduction

Dans ce chapitre, seront mis en avant quelques solutions open source de clustering, et seront explicitées.

Le choix s'est porté sur deux solutions différentes :

- La première utilise le principe de l'équilibrage de charge dans le but de la performance.
- La seconde est une librairie pour langage C ou Fortran, pour paralléliser les programmes.

L'intérêt de ce choix est d'avoir une vue sur la différences entre ces deux solutions.

3.2 OpenMosix

3.2.1 Généralités

OpenMosix [5] est une extension du noyau Linux pour mettre en œuvre un systeme à image unique pour cluster [1]. Cette extension du noyau transforme un réseau d'ordinateurs ordinaire en un super-ordinateur pour les applications Linux.

Après l'installation de openMosix, les nœuds dans le cluster communiquent entre eux, et le cluster s'adapte lui même pour la charge du travail. Les processus dans un nœud donné peuvent être migré si se nœud est chargé par rapport à un autre. OpenMosix essai continuellement d'optimiser la répartition des ressources.

Avec un patch pour le noyau Linux, nous parvenons à la création d'un système fiable,

[1] L'obejctif d'un système à image unique pour cluster (singel-system image clustering) est de paraitre comme une seule machine au niveau des utilisateurs.

rapide et économique d'un cluster qui est évolutif et adaptatif. A l'aide d'openMosix Auto Discovery, un nouveau nœud peut être ajouté quand le cluster est en marche, qui va automatiquement commencer à utiliser les nouvelles ressources.

Il n'y a aucun besoin de programmer une application spécifique pour openMosix. Puisque toutes les extensions de openMosix sont incluses dans le noyau, n'importe quelle applications Linux va automatiquement et de manière transparente bénéficier du concept de la répartition de charge de openMosix (load-balancing). Le cluster se comporte comme un multiprocesseur symétrique (SMP Symmetric Multi-Processor), et cette solution dépasse l'échelle d'un millier de nœud qui peuvent être un SMPs.

3.2.2 Fonctionnement

La technologie de openMosix[6] est constituée de deux parties : un mécanisme de migration preemptive de processus (Preemptive Process Migration PPM) et d'un ensemble d'algorithmes pour l'adaptation de partage des ressources (adaptive resource sharing algorithms). Ces deux parties sont implémentées au niveau du noyau, qui se charge en tant que module. Ainsi, elles sont totalement transparentes au niveau de l'application.

La PPM peut migrer n'importe quel processus, à n'importe quel moment, à n'importe quel nœud actif. Habituellement, les migrations sont basées sur l'information fournie par l'un des algorithmes de partage de ressources, mais les utilisateurs peuvent migrer leurs processus manuellement.

Chaque processus a son propre nœud d'origine (UHN). Normalement c'est dans ce nœud que l'utilisateur est connecté. Mais avec openMosix n'importe quel processus semble fonctionner dans son nœud d'origine (UHN), et tous les processus d'une session d'un utilisateur partagent l'environnement d'exécution du nœud d'origine (UHN). Les processus qui migrent vers d'autre nœuds (distants), utilisent les ressources locales (du nœud distant), mais reste en interaction avec l'environnement du nœud d'origine (UHN).

PPM est l'outil principale pour les algorithmes de gestion de ressource. Tant que la charge CPU et la mémoire disponible sont au dessous d'un certain seuil, les processus de l'utilisateur s'exécutent dans leur nœud d'origine. Lorsque les ressources utilisées (CPU + mémoire) dépassent un certain seuil, certains processus vont migrer vers d'autres nœuds pour tirer parti de leurs ressources disponibles. L'objectif est d'optimiser la performance par l'utilisation de l'ensemble des ressources disponible dans le réseau.

OpenMosix n'a aucun nœud central ou de relation maître/esclave entre les nœuds. Chaque nœud opère de manière autonome, et prend ses propres décisions indépendamment des autres nœuds. Cette conception permet une configuration dynamique, ou les nœuds peuvent joindre ou quitter le cluster sans l'interrompre ou le perturber. En plus, cela permet une grande extensibilité. L'extensibilité est obtenu grâce à l'intégration d'un algorithme de contrôle aléatoire, ou chaque nœud base ses décisions sur la connaissance partielle de l'état des autres nœuds et n'essaye pas de déterminer l'état de tous les nœuds

ou d'un nœud particulier du cluster. Chaque nœud envoie les informations de ses ressources disponibles à un sous-ensemble aléatoire du cluster dans un intervalle de temps régulier. Après un certain temps tous les nœuds auront les informations sur chaque nœud et gardent l'information la plus récente.

Protocole du réseau

La communication entre les nœuds se fait via les ports TCP et UDP à l'aide du protocole TCP/IP.

- Le port TCP 4660 est utilisé pour la migration des processus.
- Chaque nœud envoie son statut à une liste aléatoire d'autres nœuds pour les informer de son taux de charges en utilisant le port UDP 5428.
- Le port UDP 1334 est utilisé pour la découverte automatique des nœuds. (openMosix AutoDiscovery).
- Et le port TCP 723 est utilisé pour le systeme de fichier de openMosix (oMFS openMosix File System).

Les algorithmes de partage de ressources

Les principes des algorithmes de partage de ressources de openMosix, sont la répartition de charge et l'organisation de la mémoire. L'algorithme de répartition de charge dynamique tente continuellement de réduire la charge des différents nœuds, par la migration des processus des nœuds les plus chargés vers les nœuds les moins chargés. Le nombre de processeurs dans chaque nœud et leurs vitesses sont les facteurs importants pour l'algorithme de répartition de charge.

Le système de fichier de openMosix : MFS et DFSA

Le problème :[7]
Lorsqu'on migre sur un autre nœud, un processus qui réalise des lectures/écritures sur disque, et de faire en sorte qu'il puisse accéder à ses données sur disque sans avoir à être renvoyé sur son nœud d'origine. Si un processus crée un fichier sur le disque d'un nœud A, comment pourra-t' il encore écrire sur ce fichier du nœud A s'il est migré ailleurs, ?

MFS ou oMFS (openMosix File System) est le système de fichiers de openMosix utilisé 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 nœud de migration.

- oMFS permet d'accéder au système de fichier des autres nœuds comme s'il était monté localement. C'est un système de fichier dédié au cluster réseau qui apporte des fonctionnalités supplémentaires à NFS (Network File System). oMFS est symétrique, c'est à dire que chaque nœud voit et accède au système de fichiers des

autres nœuds. Il n'y a pas de goulot d'étranglement qui pourrait survenir avec un serveur de fichier unique centralisé. Quelque 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 nœud 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 nœud 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 nœud d'origine.

3.2.3 Les différents types de configurations

Dans le cas ou nous avons un ensemble de nœuds permanents avec openMosix qu'on va appelé serveurs, nous pouvons ajouter des stations de travail, appartenant à des utilisateurs, en les démarrant sur le noyau openMosix.

Nous pouvons envisager pour ces stations de travail, les configurations suivante :[8]

Single Pool

Dans ce cas, les stations de travail font partie du cluster, et peuvent envoyer et recevoir des processus. Nous aurons un cluster mixte.
- avantage : les stations de travail peuvent tirer parti des capacités des serveurs.
- inconvénient : les stations de travail serviront de machines de traitement.

Server Pool

Dans ce cas, les stations de travail des utilisateurs ne font pas partie du cluster. Elles n'ont pas de noyau openMosix ou n'ont pas booté dessus. Les utilisateurs doivent avoir des comptes pour se connecter sur les machines du cluster, et lancer leur taches directement sur le cluster.

Adaptive Pool

Dans cette configuration, les station de travail font partie du cluster à des moments particuliers. Par exemple, dans les moments ou un utilisateur n'utilise pas sa machine, un script lance openMosix pour que la machine rejoigne le cluster et le quitte lorsque l'utilisateur utilise sa machine. C'est l'utilisateur qui planifie ces moments d'absence (la nuit par exemple).

3.2.4 Les outils de openMosix

Les outils de openMosix permettent de visualiser certaines informations. Les principales sont :

- mosmon : moniteur openMosix. Il affiche l'état de vos nœuds openMosix y compris la charge CPU, la mémoire installée et utilisée, etc.
- mtop : Version améliorée de top qui affiche sur quel nœud les processus tournent.
- mps : Version améliorée de ps qui affiche les numéros de nœud.
- mosctl whois : Très utilise car mosmon et les autres outils n'affichent qu'un numéro de nœud.
- mosctl whois nodenumber affiche l'adresse IP ou le nom d'hôte du nœud.
- openmosixview : C'est un outil graphique de gestion pour les cluster OpenMosix. Il contient huit applications utiles pour le monitoring et l'administration. [9]

3.3 PVM

3.3.1 Génaralités

PVM (Parallel Virtual Machine) est un logiciel, développé à Oak Ridge National Laboratory, qui permet d'utiliser un ensemble de stations de travail Unix reliées par un réseau comme une machine parallèle. Cet ensemble de stations constitue votre machine parallèle virtuelle. Des problèmes de calculs intensifs peuvent ainsi être résolus de façon plus économique en utilisant les puissances et mémoires réunies de plusieurs stations.

PVM permet d'utiliser le matériel informatique existant pour résoudre des problèmes de plus grande taille avec un coût additionnel réduit. Des centaines de sites dans le monde utilisent PVM pour résoudre de gros problèmes scientifiques, industriels et médicaux. PVM est aussi largement utilisé pour l'enseignement du parallélisme. Avec ses dizaines de milliers d'utilisateurs, PVM est devenu un standard de facto dans le monde pour le calcul distribué.[10]

3.3.2 Fonctionnement

PVM comporte deux composants :

Le daemon PVM (processus pvmd3)

C'est un processus Unix qui gère les opérations des processus utilisateurs d'une application PVM, et coordonne les communications entre les machines.
Il y a un daemon par machine composant votre machine virtuelle. (Si il y a d'autres applications PVM pour d'autres utilisateurs, ils y aura d'autres daemons PVM).

Chaque daemon maintient une table de configuration de la machine virtuelle et des informations sur les processus de l'application.
Les processus utilisateurs communiquent entre eux par l'intermédiaire des daemons.
Chaque machine doit avoir sa version de pvmd3 correspondant à son architecture.

Les routines bibliothèque (libpvm3.a, libfpvm3.a et libgpvm3.a)

Généralement, ces bibliothèques sont installées dans /usr/local/PVM/pvm3/lib

libpvm3.a Bibliothèque des routines d'interface pour le langage C
- Ce sont les appels de sous programmes que le programmeur peut inclure dans son application parallèle. Ils permettent de :
  - initialiser et terminer un processus.
  - emballer, envoyer et recevoir des messages.
  - synchroniser des groupes de processus à travers des "barrières".
  - obtenir et modifier dynamiquement la configuration de la machine virtuelle.
- Ces routines ne font pas de communications directes avec les autres processus, mais envoient des commandes au daemon local duquel ils reçoivent des informations en retour.
- Des conversions de format de données (XDR) sont effectuées par défaut entre les machines d'architectures différentes.

libfpvm3.a Routines additionnelles pour utiliser PVM à partir de code Fortran.

libgpvm3.a Routines additionnelles pour utiliser les groupes de processus dynamiques.

3.4 Conclusion

Il y a plusieurs solutions de clustering open source, chacune d'elle présentant des spécifiés. Alors c'est à nous de choisir l'une de ses solutions ou des les combiner comme c'est le cas du projet Beowulf[2], afin de répondre à notre cahier de charge.

[2]Beowulf est une architecture multi-ordinateurs qui peut être utilisée pour la programmation parallèle. Il utilise des éléments comme le système d'exploitation Linux, Parallel VirtualMachine (PVM) et Message Passing Interface (MPI)

Chapitre 4

Mise en œuvre et Application

Pour l'application on a choisi d'utiliser Debian Sarge 3.1 pour les raisons suivantes :

- La stabilité du système,
- la disponibilité des paquets.

4.1 OpenMosix

4.1.1 Configuration dynamique

L'installation et la configuration de openMosix pour une utilisation dynamique est assez simple à mettre en œuvre. D'ailleurs c'est le but des concepteurs, obtenir un cluster performant sans une configuration particulière mis à part l'installation du noyau et les outils openMosix.

Prérequis

Avoir un serveur DHCP (voir B.1.1). Sinon, utiliser un routeur avec un serveur DHCP.

Installation du noyau

Récupérer le noyau 2.4.26 patché avec openMosix du site :
http ://sunsite.rediris.es/pub/mirror/clusterlinux/ .
Sinon vous pouvez le compiler. (voir A.1)

Installation des outils

[11] Pour installer les outils de openMosix, il faut :

- Récupérer les sources du noyau utilisé et le patch du noyau de openMosix, disponible dans :
  http ://www.kernel.org/pub/linux/kernel/ et
  http ://sourceforge.net/.

- Copier les sources du noyau (linux-2.4.26.tar.bz2) dans /usr/src/ anssi que le patch de openMosix (openMosix-2.4.26-1.bz2), en suite :

```
/usr/src#tar -xvjf linux-2.4.26.tar.bz2
```

- Renommer le fichier décompressé, en linux-openmosix. Etant donné que les outils de openMosix sont configurés pour accéder aux sources patché dans le répertoire /usr/src/linux-openmosix pour son installation.

```
/usr/src#mv linux-2.4.26 linux-openmosix
```

- Patcher le noyau avec openMosix

```
/usr/src/linux-openmosix#bzcat ../openMosix-2.4.26-1.bz2 | patch -p1
```

- Créer le fichier de configuration du noyau.
  Pour créer le fichier de configuration du noyau, il est possible d'utiliser la configuration existante si le noyau patché (.deb) est installé.

```
#cp /boot/config-2.4.26-om /usr/src/linux-openmosix/.config
/usr/src/linux-openmosix#make menuconfig
```

  Quitter et sauvegarder.

- Installer openmosix-tools.

  Récupérer openmosix-tools disponible dans le site :
  http ://sourceforge.net/project/showfiles.php ?group_id=46729&package_id=39627
  Copier le dans /usr/local/src/

```
/usr/local/src#tar -xvzf openmosix-tools-0.3.6-2.tar.gz
/usr/local/src#cd openmosix-tools-0.3.6-2
/usr/local/src/openmosix-tools-0.3.6-2#./configure
/usr/local/src/openmosix-tools-0.3.6-2#make
/usr/local/src/openmosix-tools-0.3.6-2#make install
```

- Charger les outils de openMosix au démarrage.

```
#update-rc.d openmosix defaults
```

  Si tous fonctionne, au démarrage apparaît :

```
openMosix configuration was successful :)
```

– Dans le cas d'utilisation de MFS, il faut ajouter dans /etc/fstab, la ligne suivante :

```
cluster     /mfs     mfs     dfsa=1     0     0
```

4.1.2 Configuration statique

La configuration statique de openMosix est intéressante dans le cas ou les machines-sont prévus pour fonctionner qu'avec openMosix.

Prérequis

Suivre les même étapes pour la configuration dynamique (voir 4.1.1).

Les fichiers de configurations

openmosix.map :
Ce fichier permet de configurer la map de votre cluster, c'est à dire les adresses ip de tous les nœuds statiques.
Voici un exemple de configuration :

```
vim /etc/openmosix.map
```

```
1 192.168.10.1 9
2 192.168.10.30 1
3 192.168.10.60 10
```

openmosix.conf :
Dans ce fichier de configuration il est possible de définir le comportement du nœud, par exemple : activer l'utilisation de MFS , activer la découverte des autres nœuds ...
Voici un exemple de configuration :

```
vim /etc/openmosix/openmosix.conf
```

```
# This file can be used to change the default behaviour of the
# openMosix startup-script.
# Force autodiscovery-daemon to start, even with a valid .map-file
AUTODISC=1
# Specify which network interface the autodiscovery-daemon should listen to
AUTODISCIF=eth0
# Set the values of /proc/hpc/admin/overheads
# OVERHEADS=
# Set the values of /proc/hpc/admin/mfscosts
# MFSCOSTS=
# Set the openMosix node-id of this node
MYOMID=2002 #propre a chaque nœud du cluster
# Set maximum number of gateways between openMosix nodes (see man setpe)
# MOSGATES=
# Pass the following value to mosctl setspeed when starting the node
# man mosctl for details
# NODESPEED=10000
# Processes are allowed to migrate to other nodes.
MIGRATE=yes
# Allow guest processes to arrive.
BLOCK=no
# Don't use MFS
MFS=yes
```

4.1.3 OpenMosixView

OpenMosixView est un outil de gestion et de contrôle d'un cluster . Il permet de se connecter aux clients, de migrer certain processus manuellement, d'arrêter ou d'activer un nœud particulier et de voir l'activité des nœuds (charge CPU, mémoire libre, processus migré ...).

Installation

Prérequis :

- Installer une interface graphique kde ou gnome ou autre.
- Installer qt , les librairies de qt , les outils de développement associés à qt avec leurs dependances, x-dev, libx11-dev et libjpeg.
- Installer la version qt associé à la version de votre openmosixview. Pour openmosixview-1.4 installer qt-x11-free-3.0.3 disponible dans le site :
  http ://www.qtsoftware.com/

```
/usr/local/src#tar -xjvf qt-x11-free-3.0.3.tar.bz2
/usr/local/src#export QTDIR=/usr/local/src/qt-x11-free-3.0.3
/usr/local/src/qt-x11-free-3.0.3#./configure
/usr/local/src/qt-x11-free-3.0.3#make
/usr/local/src/qt-x11-free-3.0.3#make install
```

Installation de openmosixview-1.4 :
Pour installer openMosixView, il suffit de suivre les étapes suivantes :

```
tar -xvzf openmosixview-1.4.tar.gz
cd openmosixview-1.4
./setup /usr/local/src/qt-x11-free-3.0.3
./configure
make
cp openmosixview/openmosixview /usr/bin
cd openmosixprocs/
cd openmosixproc
cp openmosixprocs /usr/bin
cp openmosixcollector/openmosixcollector/openmosixcollector /usr/bin
cp openmosixanalyzer/openmosixanalyzer/openmosixanalyzer /usr/bin
cp openmosixhistory/openmosixhistory/openmosixhistory /usr/bin
cp openmosixcollector/openmosixcollector.init /etc/init.d/openmosixcollector
```

Ensuite, copier le binaire openmosixprocs dans /usr/bin sur tous les nœuds du cluster.
Le lancement de openmosixview s'effectue dans une console.

PS :
- OpenMosixView nécessite le droit admin, et d'utiliser ssh sans mot de passe (voir C.2).
- Pour utiliser ssh avec openmosixview, cocher use ssh dans la fenêtre de openmosixview et sauvgarder la configuration.

4.2 PVM

Contrairement à openMosix, pvm nécessite un nœud maître qu'on appellera serveur et des nœuds esclaves qu'on appellera clients.

Pvm est disponible dans le site : http ://www.netlib.org/pvm3/

4.2.1 Installation

Pour l'installer, suivre les étapes suivantes :

1. Définir les variables d'environnement, et ajouter le contenu du pvm3/lib/bashrc.stub dans $HOME/.bashrc
   Au finale ajouter à .bashrc :

```
export PVM_ROOT=$HOME/pvm3
export PVM_RSH=/usr/bin/ssh


#
# append this file to your .bashrc to set path according to machine
# type. you may wish to use this for your own programs (edit the last
# part to point to a different directory f.e. ~/bin/_$PVM_ARCH.
#
if [ -z $PVM_ROOT ]; then
if [ -d ~/pvm3 ]; then
export PVM_ROOT=~/pvm3
else
echo "Warning - PVM_ROOT not defined"
echo "To use PVM, define PVM_ROOT and rerun your .bashrc"
fi
fi

if [ -n $PVM_ROOT ]; then
export PVM_ARCH=`$PVM_ROOT/lib/pvmgetarch`
#
# uncomment one of the following lines if you want the PVM commands
# directory to be added to your shell path.
#
export PATH=$PATH :$PVM_ROOT/lib # generic
# export PATH=$PATH :$PVM_ROOT/lib/$PVM_ARCH # arch-specific
# # uncomment the following line if you want the PVM executable directory
# to be added to your shell path.
#
export PATH=$PATH :$PVM_ROOT/bin/$PVM_ARCH
fi
```

2. Pour utiliser ssh, modifier la ligne usr/bin/rsh en usr/bin/ssh dans le fichier /pvm3/conf/LINUX.def
3. Au finale, un make pour l'installer.

4.2.2 Les fichiers de configurations

Les fichiers de configuration de pvm permettent d'ajouter d'autre nœuds et de lancer le daemond pvmd dans ces nœuds. Pour cela il faut :

- Installer pvm dans tous les nœuds du cluster.
- Le fichier /etc/hosts doit être bien configuré. (voir B.4).
- Configurer ssh pour qu'il se connecte avec un nom. (voir C.3).
- Pour ne pas utiliser de mot de passe avec ssh voir C.2.
- Créer le fichier config-pvm

```
vim $HOME/pvm3/lib/config-pvm
```

```
* ep=$PVM_ROOT/bin/$PVM_ARCH
&client01 dx=/home/client01/pvm3/lib/pvmd #chemain vers pvmd pour client01
&client02 dx=/home/client01/pvm3/lib/pvmd
```

– Le répertoire de travail ($HOME/pvm3/bin/LINUX) doit être partagé avec tous les nœuds via NFS (voir B.3), sans oublier de monter les répertoires dans /etc/fstab pour les clients (voir l'exemple dans B.4).

4.3 Tests et résultats

Pour évaluer la performance des solutions mises en œuvre, nous avons effectué des tests sur un produit matricielle avec différentes dimensions. Ces derniers ont été effectué sur un cluster avec un nombre de nœuds variant.

4.3.1 Machines homogènes

Tests

Dans cette partie de tests, nous avons utiliser des machines esclaves avec la configuration suivante :Fujitsu Siemens, Intel Pentium IV 2.66 GHz , RAM 256 Mo. (Voir illustration figure 4.1).

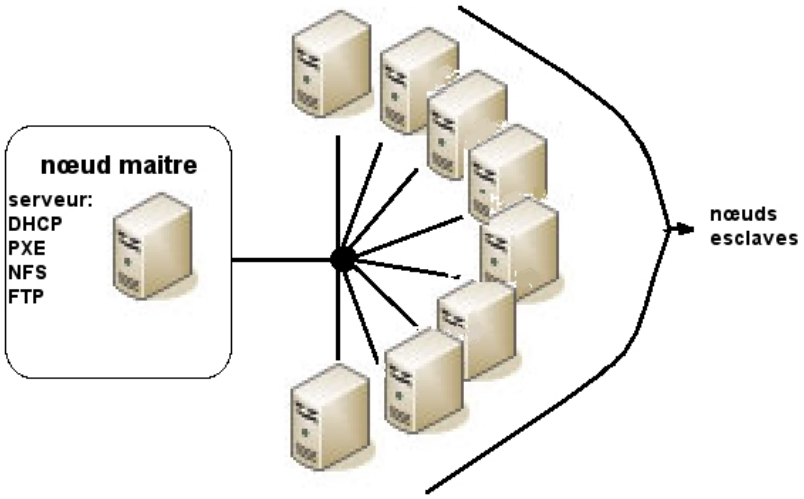


FIG. 4.1 – Illustration de la configuration utiliser pour les tests sous pvm

la première étape est le lancement du programme test sur un cluster configuré avec PVM dont le code source est disponible dans l'annexe D.1.

L'organigramme du programme de test est illustré dans la figure 4.2.

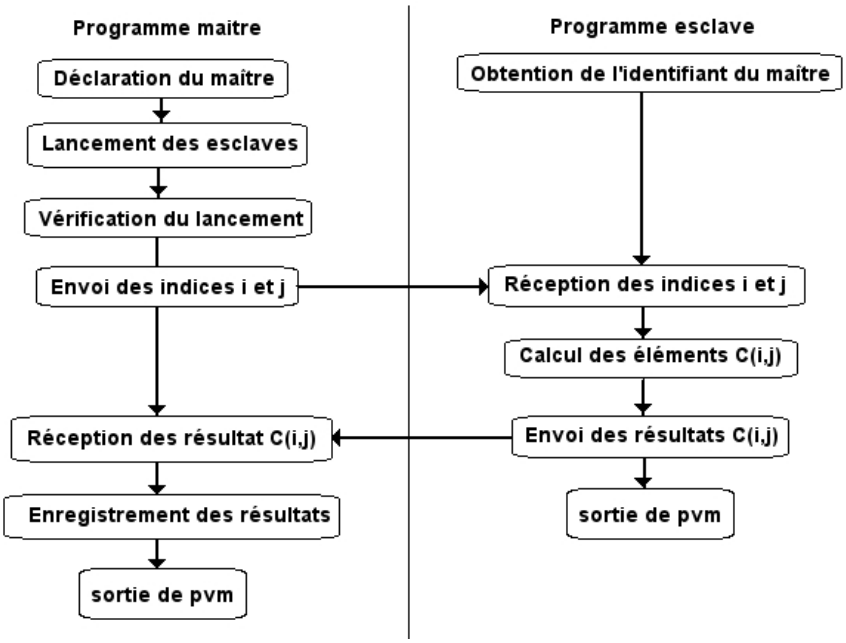


FIG. 4.2 – Organigramme du programme de test sous pvm

Les résultats des tests sur un cluster homogène sont représentés dans les graphes suivants : (voir figures 4.3, 4.4, 4.5 et 4.6)

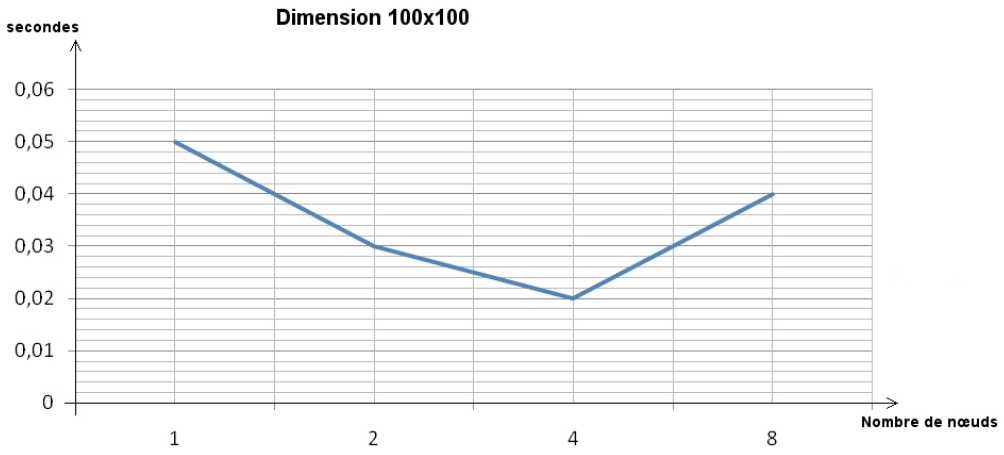


FIG. 4.3 – Résultat d'un produit matricielle de dimesion 100x100

Dans le seconde étape, les même résultats sont présent pour un cluster de 8 noeuds, avec openmosix associé à pvm.

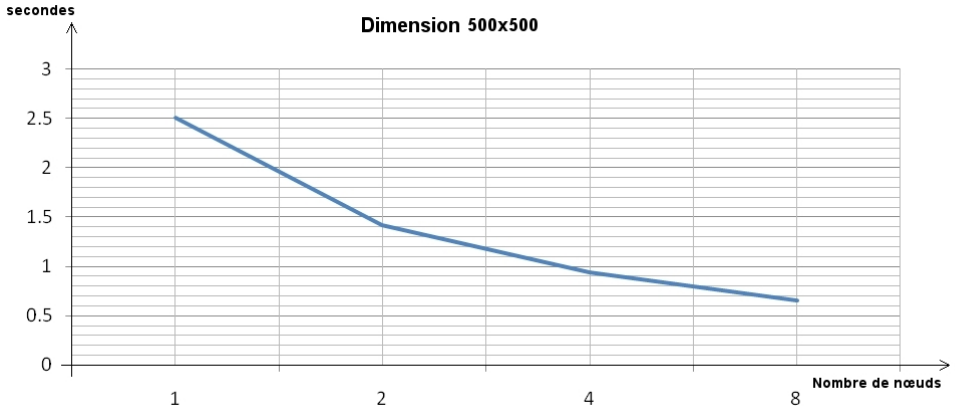


FIG. 4.4 – Résultat d'un produit matricielle de dimesion 500x500

FIG. 4.5 – Résultat d'un produit matricielle de dimesion 1000x1000

Commentaires

Pour la figure 4.3, on constate que la performance augmente avec le nombre d'esclaves ; mais des qu'on dépasse 4 esclaves, ça engendre une dégradation de performance. Cette dernière est due au pertes de temps dans la communication, qui devient plus long par rapport à la durée du calcul.

Pour les figures 4.4, 4.5 et 4.6, on constate que la performance augmente avec le nombre d'esclaves. Cette augmentation est presque linaire dans une dimension grande (par exemple 1023x1023) parce que le temps de communication est pratiquement négligeable par rapport à la durée de calcul.

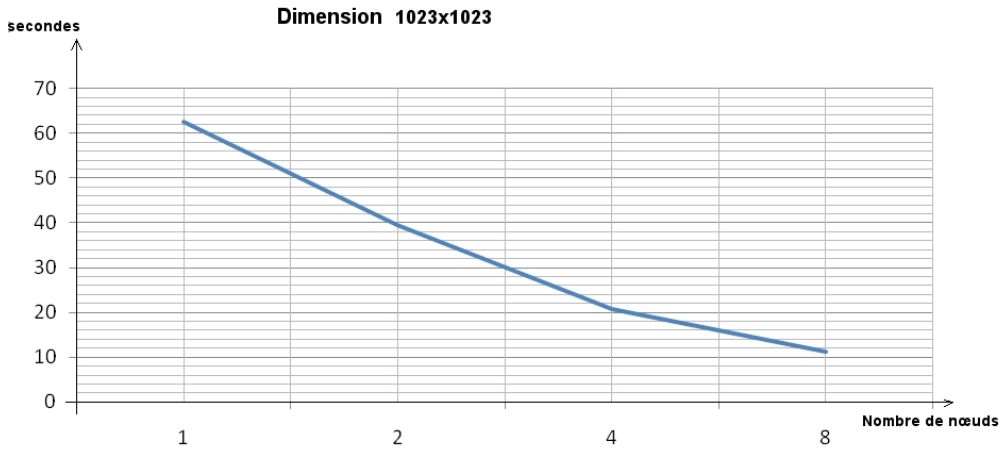


FIG. 4.6 – Résultat d'un produit matricielle de dimesion 1023x1023

4.3.2 Machines hétérogènes

Tests

Dans cette partie de tests, nous avons utiliser des machines esclaves avec les configurations suivantes : 5 PC Fujitsu Siemens, CPU Intel Pentium IV 2.66 GHz , RAM 256 Mo et 3 PC Foxconn, CPU Intel EM64T, RAM 512 Mo. (Voir illustration figure 4.1).

Le lancement du programme test sur un cluster configuré avec PVM dont le code source est disponible dans l'annexe D.1 donne le résultat représenté dans le graphe 4.7

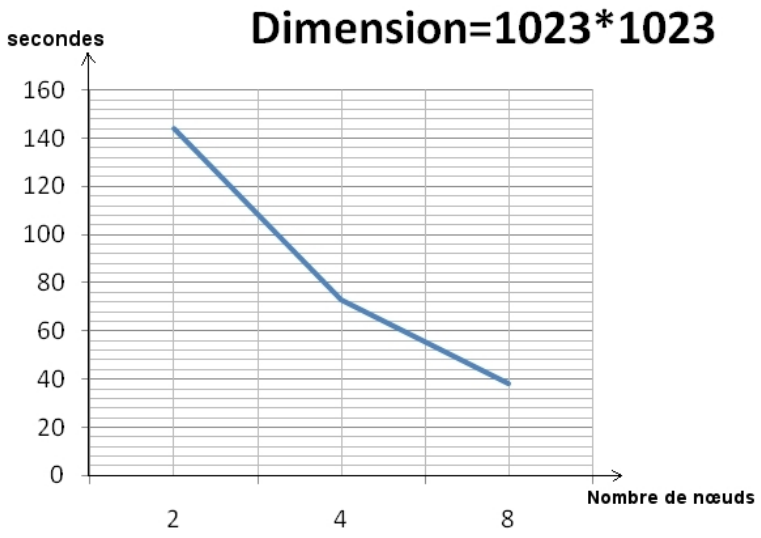


FIG. 4.7 – Résultat d'un produit matricielle de dimesion 1023x1023

Commentaires

Dans ce cas, on remarque que la performance à diminuer par raport à un cluster homogène. Et cela est due à l'intruduction des machines moins performantes (Foxconn). Ces dernière retardent les machines performantes.

4.3.3 Stress test pour openMosix

Dans cette partie, nous avons mis au point un test pour évaluer et démontrer l'intérêt de openMosix. Ce test consiste à lancer un script plusieurs fois, sur un cluster de 2 nœuds. Le code source est disponible dans l'annexe D.2
La configuration du cluster est hétérogène (voir l'illustration dans la figure 4.8). Elle se compose de :

- ASUS : Intel Pentium, CPU core Duo 1.73 Ghz, RAM 2 Go.
- ASUS : CPU AMD64 3000+ (1.8 GHz), RAM 1Go.

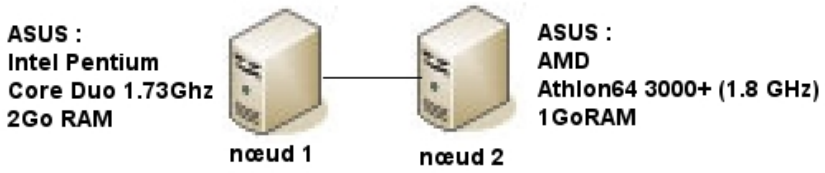


FIG. 4.8 – Illustration de la configuration utiliser pour les tests sous OpenMosix

Les résultats sont disponible dans la figure 4.9

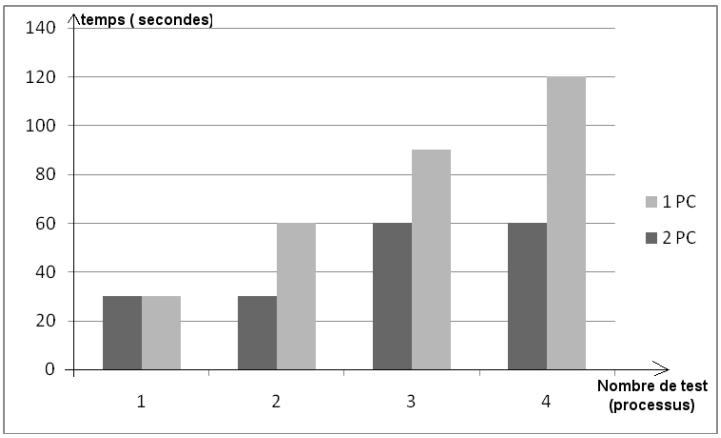


FIG. 4.9 – Illustration de l'équilibrage de charge sous openMosix

Commetaire

On remarque que le temps d'exécution d'un processus est de 30 secondes. Le lancement d'un processus sur un cluster de deux nœuds ne change rien au résultat ; car OpenMosix travail à l'échelle du processus.

Le lancement de deux processus sur un cluster de deux nœuds permet de minimiser le temps d'exécution par rapport à une machine ; car OpenMosix répartie la charge sur les deux nœuds. Donc chaque nœud exécute un processus, et on peut gagné jusqu'à 50% du temps d'exécution que sur une seul machine.

Le lancement de trois processus sur un cluster de deux nœuds permet de gagné jusqu'à 25% du temps d'execution ; car OpenMosix répartie deux processus sur un nœud et un processus sur l'autre nœud.

Conclusion

- L'utilisation de openMosix sur notre configuration, permet de gagner du temps comparativement à une seule machine.
- OpenMosix s'arrange de distribuer les processus sur l'ensemble des nœuds de façon à ce que la charge totale sur le nœud d'origine soit bien répartie
- Il n'est pas nécessaire d'avoir autant de nœuds que de processus car openMosix travaille à l'échelle d'un processus.

4.4 Conclusion

De cette étude, on remarque bien la différence entre pvm et openMosix, qui réside dans les besoins de l'application.

OpenMosix est plus orienté pour supporter une charge importante sur le système, qui se compose de plusieurs processus. Le matériel utilisé dans ce cas, n'est pas forcément performant et peut être hétérogène.

Par contre, pvm est orienté pour le calcul. Il doit utiliser des machines performantes et homogènes avec une connexion Gigabite de préférence, pour minimiser le temps de communication et de calcul. La présence d'un élément moins performant influe négativement sur le résultat.

Chapitre 5

Conclusion et perspectives

5.1 Conclusion

Notre projet de fin d'étude s'est porté essentiellement sur la mise en œuvre d'une solution de clustering open source. L'intérêt d'une telle installation réside dans le fait que la mise en place de la solution est dépendante du cahier de charge qu'on fixe au préalable. En plus, les logiciels du clustering sont disponibles sur le réseau internet ainsi que leurs quasi-gratuités.

Dans notre cas, on a choisit deux solutions différentes : OpenMosix et PVM. On a bien constaté que chacune de ces solutions présente des avantages et des inconvénients.

En ce qui concerne OpenMosix, son principal avantage est l'équilibrage de charge, il permet de réguler la charge utilisée dans le cluster et de la répartir de manière transparente. Néanmoins l'inconvénient qui se pose est que sont développement s'est arrété et il n'est pas adapter au dernières avancées Linux.

En ce qui concerne PVM, rappelons-nous que c'est une librairie de programmation parallèle en C. Elle permet de répartir une tâche importante sur les nœuds du cluster et de gagner en performance. La performance d'un cluster avec PVM évolue de manière non linéaire. Au bout d'un certain nombre de nœuds, les performances commencent à se dégrader. La dégradation se présente lorsque le temps de calcul est inférieur au temps que le système mets à envoyer les requêtes de calculs et les récupérer.

Finalement, pour réaliser un cluster, il faudra établir un cahier de charge bien défini. Avec la connaissance de ce dernier, on pourra choisir le matériel nécessaire ainsi que les outils qui permettent sa mise en œuvre.

5.2 Perspectives

Avec les avancées technologiques dans le domaine des réseaux, on est arrivé à des liaisons à plus d'un gigabits, ce qui permet de dire que les communications sont quasi instantanées. Avec le travail que nous avons effectué dans le cadre de notre projet de fin d'étude, on peut le transposer et l'insérer dans le cadre d'un circuit spécialisée FPGA par exemple et de fabriquer un système multiprocesseur en utilisant les techniques de clustering.

Le recours à des circuits spécialisés dont l'architecture est optimisée pour la réalisation de certaines fonctions constitue aussi une solution à explorer pour gagner en performance. Dans le cas des systèmes embarqués, ce gain de performance pourra résulter de l'intégration fine au sein d'un même circuit des divers composants, matériels et logiciels, pour former ce qu'il convient d'appeler un SOC (System On Chip). Ces divers composants peuvent par ailleurs être regroupés en réseau au sein d'un même circuit (Network On Chip).

Annexe

Annexe A

Compilation du noyau

A.1 Noyau patché avec openMosix

Pour compiler un noyau patché avec openMosix, suivre les étapes suivante :

- Installer les outils nécessaire à la compilation.

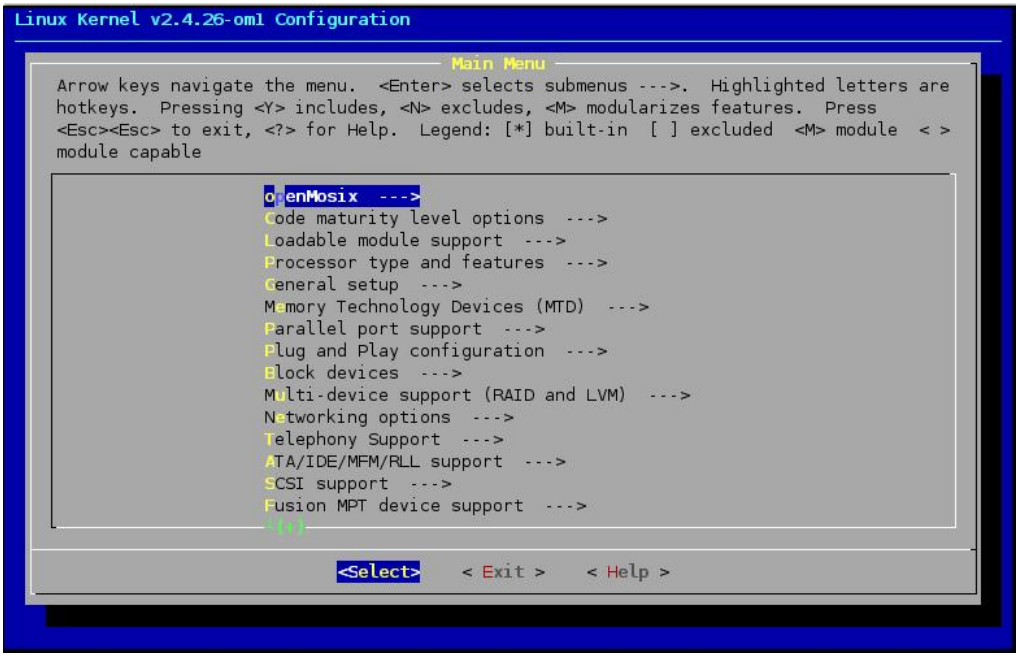
```
#apt-get install debconf-utils dpkg-dev debhelper build-essential kernel-package
libncurses-dev unzip bzip2
```

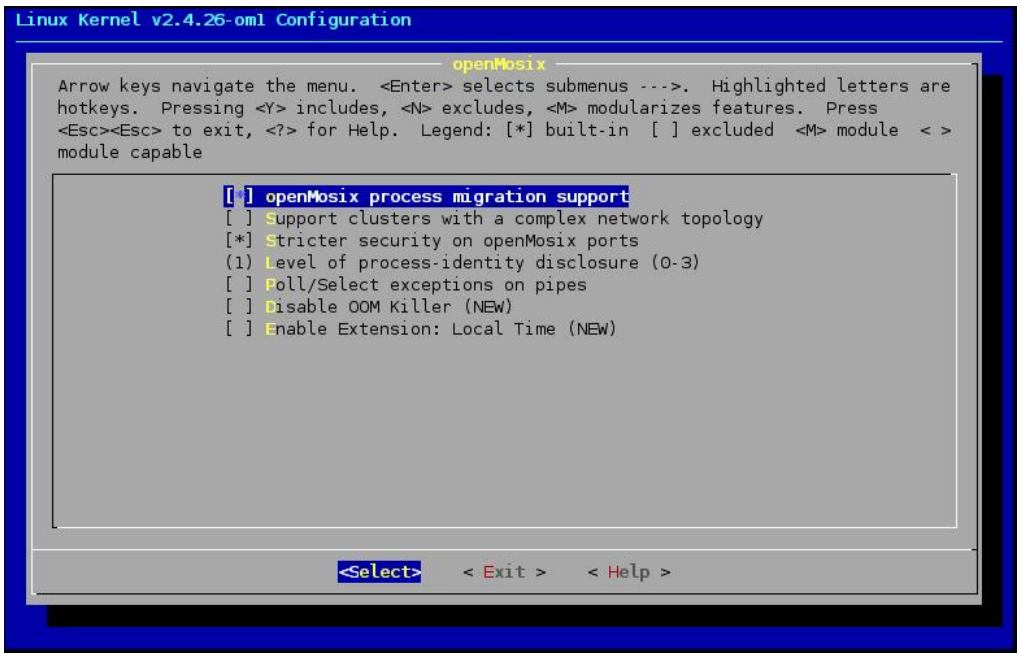
- Récupérer le noyau dans le site :
  http ://www.kernel.org/pub/linux/kernel/
  et le patch openMosix correspondant au noyau, dans le site :
  http ://sourceforge.net/project/showfiles.php ?group_id=46729 .
  Copier les dans : /usr/src/

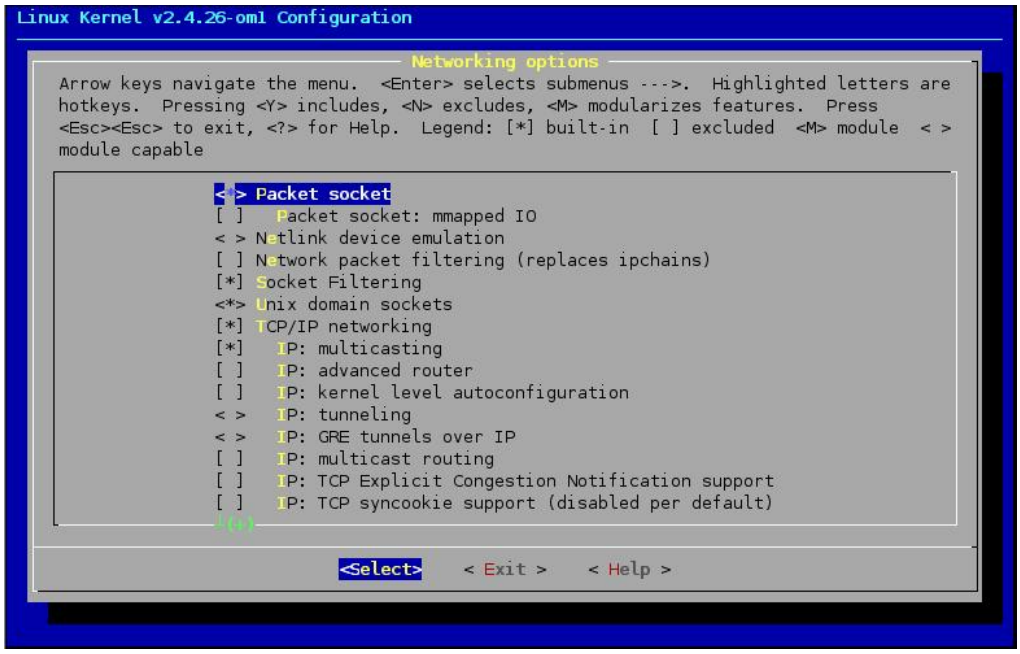
-
```
/usr/src#tar -xvjf linux-2.4.26
/usr/src/linux-2.4.26#bzcat ../openMosix-2.4.26-1.bz2 | patch -p1
```

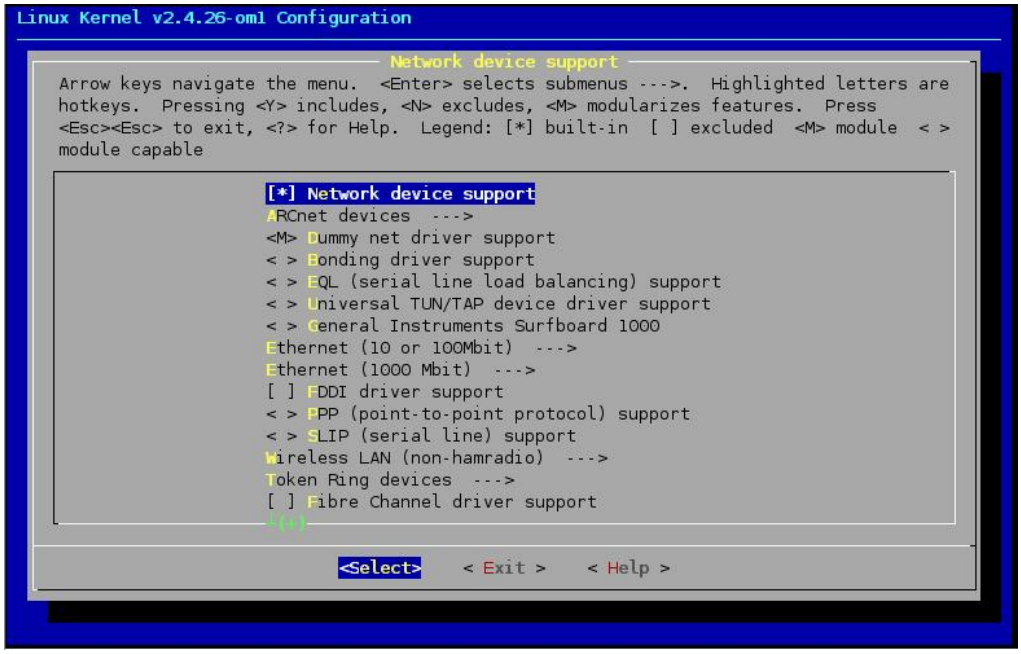
- Configuration du noyau :

```
/usr/src/linux-2.4.26#make menuconfig
```

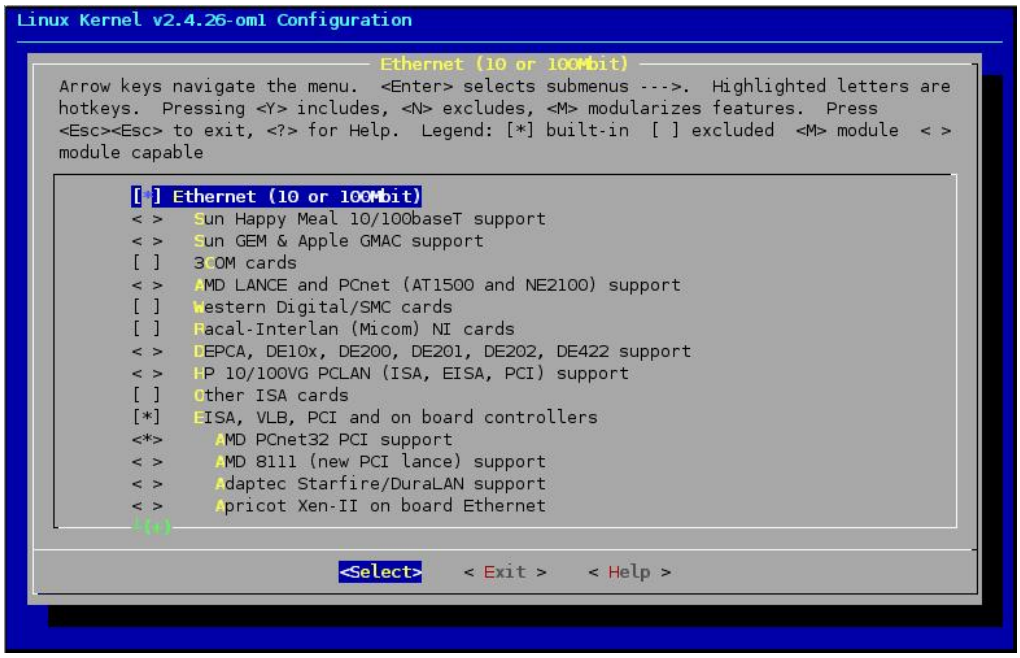

Linux Kernel v2.4.26-om1 Configuration
Main Menu
Arrow keys navigate the menu. <Enter> selects submenus --->. Highlighted letters are hotkeys. Pressing <Y> includes, <N> excludes, <M> modularizes features. Press <Esc><Esc> to exit, <?> for Help. Legend: [*] built-in [ ] excluded <M> module < > module capable
openMosix --->
Code maturity level options --->
Loadable module support --->
Processor type and features --->
General setup --->
Memory Technology Devices (MTD) --->
Parallel port support --->
Plug and Play configuration --->
Block devices --->
Multi-device support (RAID and LVM) --->
Networking options --->
Telephony Support --->
ATA/IDE/MFM/RLL support --->
SCSI support --->
Fusion MPT device support --->
<Select> < Exit > < Help >


Linux Kernel v2.4.26-om1 Configuration
openMosix
Arrow keys navigate the menu. <Enter> selects submenus --->. Highlighted letters are hotkeys. Pressing <Y> includes, <N> excludes, <M> modularizes features. Press <Esc><Esc> to exit, <?> for Help. Legend: [*] built-in [ ] excluded <M> module < > module capable
[*] openMosix process migration support
[ ] Support clusters with a complex network topology
[*] Stricter security on openMosix ports
(1) Level of process-identity disclosure (0-3)
[ ] Poll/Select exceptions on pipes
[ ] Disable OOM Killer (NEW)
[ ] Enable Extension: Local Time (NEW)
<Select> < Exit > < Help >





Cocher la carte réseau correspondante à votre matériel.
Pour un noyau compatible avec n'importe quel carte réseau, il faut coher tout



Quiter et sauvegarder.

– Compilation.

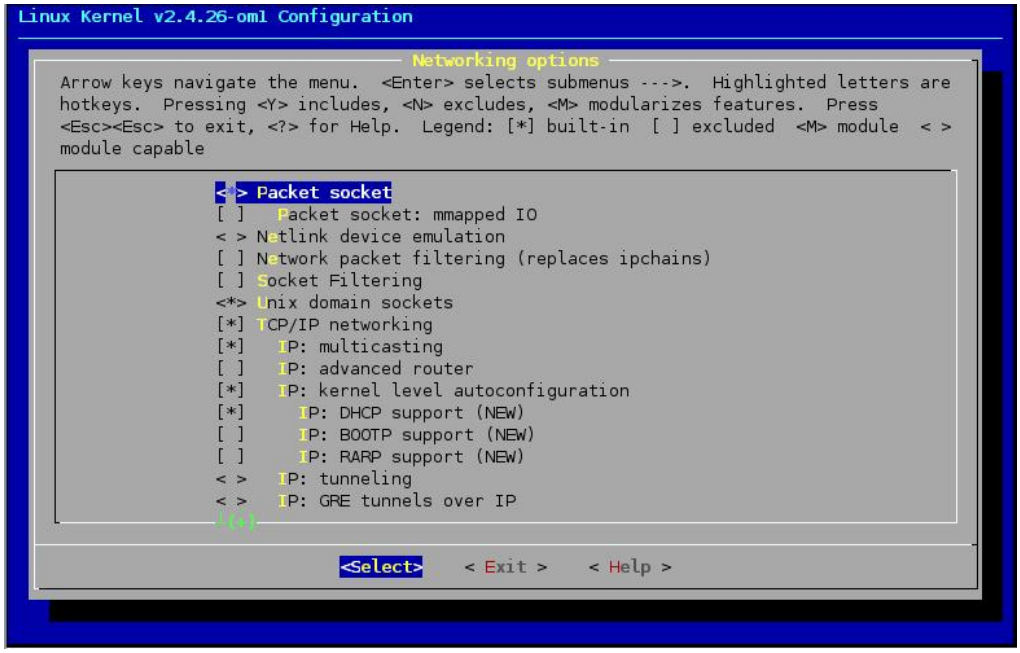
```
make dep

make clean bzImage modules modules_install
```

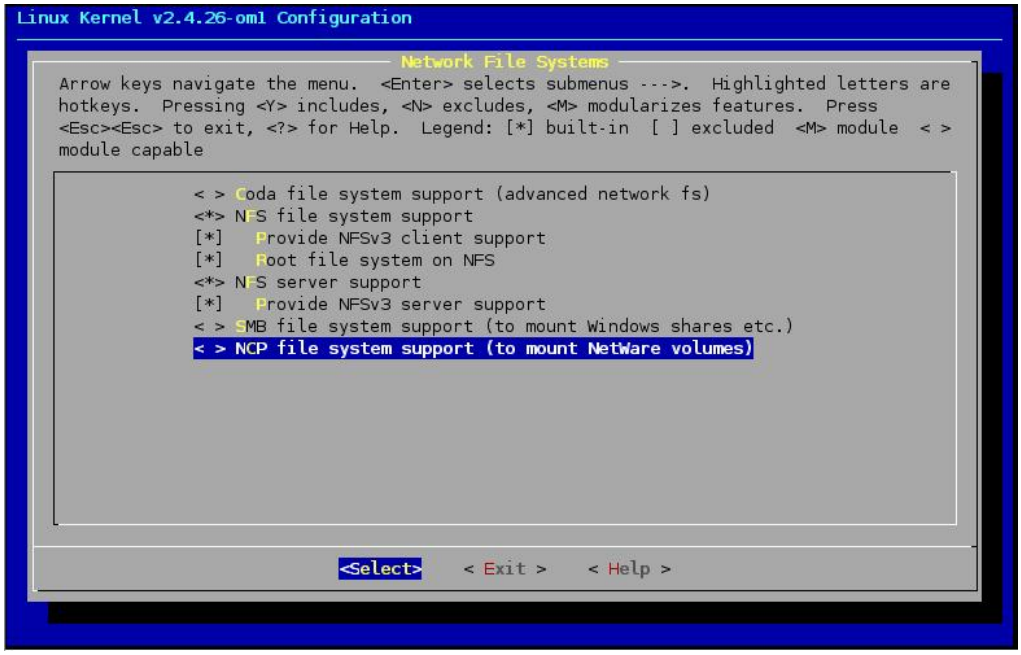
– Utiliser le noyau au démarrage.

```
#cp /usr/src/linux-2.4.26/arch/i386/boot/bzImage

/boot/vmlinuz-2.4.26-om

#cp /usr/src/linux-2.4.26/System.map /boot/System.map-2.4.26-om

#cp /usr/src/linux-2.4.26/.config /boot/config-2.4.26-om

#update-grub
```

A.2 Noyau pour clients sans disque

Suivre les même étapes décrites dans A.1, et changer : Networking options et Network File System comme ce qui suit :


Linux Kernel v2.4.26-om1 Configuration
Networking options
Arrow keys navigate the menu. <Enter> selects submenus --->. Highlighted letters are hotkeys. Pressing <Y> includes, <N> excludes, <M> modularizes features. Press <Esc><Esc> to exit, <?> for Help. Legend: [*] built-in [ ] excluded <M> module < > module capable
<*> Packet socket
[ ] Packet socket: mmapped IO
< > Netlink device emulation
[ ] Network packet filtering (replaces ipchains)
[ ] Socket Filtering
<*> Unix domain sockets
[*] TCP/IP networking
[*] IP: multicasting
[ ] IP: advanced router
[*] IP: kernel level autoconfiguration
[*] IP: DHCP support (NEW)
[ ] IP: BOOTP support (NEW)
[ ] IP: RARP support (NEW)
< > IP: tunneling
< > IP: GRE tunnels over IP
<Select> < Exit > < Help >


Linux Kernel v2.4.26-om1 Configuration
Network File Systems
Arrow keys navigate the menu. <Enter> selects submenus --->. Highlighted letters are hotkeys. Pressing <Y> includes, <N> excludes, <M> modularizes features. Press <Esc><Esc> to exit, <?> for Help. Legend: [*] built-in [ ] excluded <M> module < > module capable
< > Coda file system support (advanced network fs)
<*> NFS file system support
[*] Provide NFSv3 client support
[*] Root file system on NFS
<*> NFS server support
[*] Provide NFSv3 server support
< > SMB file system support (to mount Windows shares etc.)
< > NCP file system support (to mount NetWare volumes)
<Select> < Exit > < Help >

Annexe B

Les serveurs

B.1 Serveur DHCP

B.1.1 Serveur DHCP pour un réseau local

– Installer le serveur DHCP.

```
#apt-get install dhcp3-server
```

Dans notre cas, c'est eth0.

La configuration peut se refaire avec :

```
#dpkg-regonfigure dhcp3-server
```

– Configurer la carte réseau en statique afin que le serveur DHCP puisse démarré.

```
#vi /etc/network/intefaces
auto lo
iface lo inet loopback
#
auto eth0
iface eth0 inet static
address 192.168.0.1
netmask 255.255.255.0
```

Redémarrer la carte réseau.

```
#/etc/init.d/networking restart
```

– Configurer le serveur DHCP.

```
#vi /etc/dhcp3/dhcpd.conf
ddns-update-style none ;
option domain-name "om.cluster.cal" ;
deafault-lease-time 3600 ;
max-lease-time 7200 ;
authoritative ;
log-facility local7 ;
subnet 192.168.0.0 netmask 255.255.255.0 {
range 192.168.0.2 192.168.0.254 ;
}
```

Lancer le serveur DHCP.

```
#/etc/init.d/dhcp3-server start
```

B.1.2 Serveur DHCP pour clients PXE

Pour configurer un serveur dhcp pour clients pxe, suivre les même étapes de la configuration que pour un serveur dhcp pour un réseau locale (voir B.1.1). La différence est dans le fichier de configuration dhcpd.conf qui doit être comme suit :

```
#vi /etc/dhcp3/dhcpd.conf
```

```
ddns-update-style none;
option domain-name "om.cluster.cal";
#
deafault-lease-time 86400;
max-lease-time 86400;
#
authoritative;
#
log-facility local7;
#
subnet 192.168.10.0 netmask 255.255.255.0 {
range 192.168.10.2 192.168.10.254;
option routers 192.168.10.1; #addresse de la passerelle
option broadcast-address 192.168.10.255;
server-name "om"; #nom du serveur
filename "pxelinux.0";
}
host client01 {
hardware ethernet 00:1c:25:3a:49:22;
fixed-address 192.168.10.2;
}
```

B.2 Serveur FTP

La configuration d'un serveur ftp est simple, pour cela il suffit d'installer tftp-hpa et le configurer comme suit :

```
#apt-get install tftp-hpa
```

```
vim /etc/defaults/tftp-hpa
```

```
RUN_DAEMON="yes"

OPTION="-l -s /diskless" #repertoire à partager
```

B.3 Serveur NFS

```
#apt-get install nfs-kernel-server
```

```
#vim /etc/default/nfs-common
```

```
STATDOPTS="-port 32765 -outgoing-port 32766"
NEED_LOCKD=
```

```
#vim /etc/default/nfs-kernel-server
```

```
#Number of servers to start up
RPCNFSDCOUNT=8
#Options for rpc.mountd
RPCMOUNTDOPTS="-p 32767"
```

```
#vim /etc/modprobe.d/lockd
```

```
options lockd nlm_udpport=32764 nlm_tcpport=32764
```

Après cela il ne reste qu'a mettre les dossier à partager dans le fichier /etc/exports. Voici un exemple pour le partage des dossier pour des clients sans disque.

```
#vim /etc/exports
```

```
# /etc/exports : the access control list for filesystems which may be exported
# to NFS clients. See exports(5).
#
/diskless/client01/nfsroot 192.168.10.2(rw,sync,no_root_squash,subtree_check)
/diskless/client02/nfsroot 192.168.10.3(rw,sync,no_root_squash,subtree_check)
#
/home/om/pvm3/bin/LINUX client01(rw,sync,no_root_squash,subtree_check)
/home/om/pvm3/bin/LINUX client02(rw,sync,no_root_squash,subtree_check)
#
/usr client01(rw,sync,no_root_squash,subtree_check)
/usr client02(rw,sync,no_root_squash,subtree_check)
#
/opt client01(rw,sync,no_root_squash,subtree_check)
/opt client02(rw,sync,no_root_squash,subtree_check)
```

B.4 Serveur PXE

Prérequis

- Installer un serveur dhcp (voir B.1.2).
- Installer un serveur tftp (voir B.2).
- Installer un serveur nfs (voir B.3).
- Préparer un noyau pour les clients sans disque (voir A.2).

– Installer syslinux et pxe.

```
#apt-get install syslinux pxe
```

Une fois tous les prérequis installés et configurés, préparer les clients pxe en suivant les étapes suivantes :

– Créer le répertoire d'un client dans le répertoire partagé par le serveur ftp, dans lequel il faut mettre les fichiers/dossiers de base que linux utilise. Dans notre exemple, le répertoire du serveur ftp se nome diskless, et le répertoire du client se nome client01.

– Copier le noyau prévu pour les clients sans disque dans le répertoire du client.

```
#cp vmlinuz-2.version /diskless/client01
```

– Créer le répertoire racine dans le répertoire du client qu'on nommera nfsroot dans notre exemple.

```
#mkdir /diskless/client01/nfsroot
```

– Copier les répertoires nécessaire au démarrage du client dans le répertoire du client. Les répertoires obligatoires sont : /etc /dev /bin /lib /sbin /root . Ces repertoires sont propres pour chaque client.
Les autre répertoires /usr /opt /home peuvent être partager avec le serveur et avec les autres clients.
Créer les répertoires /proc /sys /media /mnt /tmp /var /var/tmp /var/run /var/lock et /mfs (dans le cas ou vous utilisez openMosix). Ces répertoires seront vides et propres à chaque client.

– Copier pxe dans le répertoire du serveur ftp. Il sera utilisé par tous les clients.

```
#cp /usr/lib/syslinux/pxelinux.0 /diskless
```

– Créer le répertoire pxelinux.cfg dans lequel il faut mettre la configuration des clients.

```
#mkdir /diskless/pxelinux.cfg
```

Le fichier de configuration du client doit avoir le nom de l'adresse ip voulu converti en hexadécimale sans les points. Par exemple pour l'adresse 192.168.10.2 vous aurez C0A80A02 . Sans oublier de donner l'adresse ip voulu au client suivant son adresse MAC dans le serveur dhcp.
Dans notre exemple, le fichier de configuration qui va permettre de lancer le client01 avec l'adresse 192.168.10.2 est le suivant :

```
#/diskless/pxelinux.cfg#vim C0A80A02
```

```
DEFAULT linux
LABEL linux
KERNEL client01/vmlinuz-2.version
APPEND ip=dhcp root=/dev/nfs nfsroot=192.168.10.1 :/diskless/client01/nfsroot
nfsopts="hard,intr" vga=normal pnpbios=off rw -
```

PS : l'adresse 192.168.10.1 est celle du serveur.

– La configuration de pxe. Modifier dans le fichier de configuration de pxe que les lignes suivantes :

```
#vim /etc/pxe.conf
```

```
interface eth0 #l'interface utiliser pour pxe
default_address=192.168.10.1 #l'adresse de votre serveur pxe
mtftp_address=192.168.10.1
tftpdbase=/diskless #répertoire partager par le serveur ftp
domain=om.cluster.cal
```

A cette étape, les clients sont presque prêts à l'emploi, il ne reste qu'à modifier les fichiers de configurations qui se trouve dans /etc (propre à chaque client).

– Modifier etc/openmosix.map et etc/openmosix/openmosix.conf dans le cas de l'utilisation de openMosix. (voir 4.1.2).

– Modifier etc/hosts lors d'une configuration statique.

```
#vim /diskless/client01/nfsroot/etc/hosts
```

```
# enlever le nom de la machine associée a l'adresse de bouclage pour ne pas avoir un
problème avec pvm
127.0.0.1    localhost.localdomain localhost
192.168.10.1  om.om.cluster.cal om
192.168.10.2  client01
192.168.10.3  client02
#
# The following lines are desirable for IPv6 capable hosts
: :1 ip6-localhost ip6-loopback
fe00 : :0 ip6-localnet
ff00 : :0 ip6-mcastprefix
ff02 : :1 ip6-allnodes
ff02 : :2 ip6-allrouters
ff02 : :3 ip6-allhosts
```

– Modifier le nom de la machine dans etc/hostname

– Modifier etc/network/interfaces

```
vim /diskless/client01/nfsroot/etc/network/inetfaces
```

```
auto lo
iface lo inet loopback
iface eth0 inet dhcp
```

– Modifier etc/fstab pour avoir :

```
# /etc/fstab : static file system information.
#
# <file system> <mount point> <type> <options> <dump> <pass>
proc         /proc     proc    defaults    0     0
/dev/nfs    /          nfs    defaults    0    0
none        /tmp      tmpfs    defaults    0    0
none        /var/tmp  tmpfs    defaults    0    0
none        /var/run  tmpfs    defaults    0    0
none        /var/lock  tmpfs    defaults    0    0
om :/usr    /usr        nfs    defaults    0    0
om :/opt    /opt        nfs    defaults    0    0
#PVM
om :/home/om/pvm3/bin/LINUX  /home/om/pvm3/bin/LINUX  nfs  defaults  0  0
#MFS
cluster     /mfs    mfs    dfsa=1    0    0
```

Finalement, il ne reste qu'à redémarrer les serveurs modifiés et lancer les clients.

Annexe C

Sécurité

C.1 La commande sudo

Pour avoir le droit d'utiliser la commande sudo, installer le paquet sudo et ajouter le nom d'utilisateur dans le fichier /etc/sudoers

```
#chmod a+w /etc/sudœrs
#vi /etc/sudœrs
```

```
root    ALL=(ALL) ALL
user    ALL=(ALL) ALL
```

```
#chmod a-w /etc/sudœrs
```

C.2 SSH sans mot de passe

Le nœud de contrôle ou de gestion doit accéder au clients via ssh. Pour simplifier la tache on autorise l'accès sans mot de passe, sans négliger la sécurité. Dans ce cas, il faut générer une clé publique et la copier dans tous les clients dans le répertoire de ssh de l'utilisateur ou du root.

- Pour générer la clé :

  ```
  ssh-keygen -t dsa
  ```

- Copier la clé publique vers tous les clients en modifiant son nom pour avoir l'accès.

  ```
  scp $HOME/.ssh/id_dsa.pub client01@192.168.10.2 :/home/client01/.ssh/authorized_keys
  ```

  Dans le cas de root

  ```
  scp /root/.ssh/id_dsa.pub root@192.168.10.2 :/root/.ssh/authorized_keys
  ```

PS : Pour openMosix on doit accéder en tant que root et pour pvm on doit accéder en tant qu'utilisateur.

C.3 Se connecter avec ssh via un nom

Dans certain cas, comme pour pvm, on doit se connecter via ssh à un client avec un utilisateur particulier, et sans spécifier le nom de l'utilisateur avec son adresse ip. Pour cela, il faut créer un fichier config dans le répertoire .ssh dans le quel on va donner les informations nécessaires.

```
vim $HOME/.ssh/config
```

```
Host om
hostname om
user om
#
Host client01 #nom de connexion
hostname machine01 # ou son adresse ip
user client01
#
Host om3
hostname machine03
user client03
```

Pour se conecter au client03, il suffit de faire :

```
ssh om3
```

Annexe D

Code source

D.1 Produit matricielle avec pvm

```
/*common.h*/

/*constante communes*/

#ifndef COMMON_H

#define COMMON_H

#define num_slaves 1

#define dim 1024

#define num_elm dim*dim

#define num_send dim*dim/num_slaves

#endif
```

```
/*tags.h*/

/*definition des etiquette utiliser pour les messages*/

#ifndef TAGS_H
#define TAGS_H
#define msg_ind 240 /*envoi du maitre vers l'esclave*/

#define msg_result 250 /*envoi de l'esclave vers le maitre*/

#define msg_time 260 /*envoi de la duree de l'esclave vers le maitre*/

#endif
```

```
    /*programme maitre*/
#include<stdio.h>
```

```
#include<stdlib.h>
#include<time.h>
#include"pvm3.h"
#include"tags.h"
#include"common.h"
int get_slave_no(int*,int);
int main()
{
float clk_sec=CLOCKS_PER_SEC;
float temps,ts;
float time[num_slaves]; /*tableau qui contient la duree d'execution des esclave*/
clock_t debut,fin;
debut=clock(); /*debut de la mesure de la duree d'execution*/
int i,j,k,l;
int mytid; /*identifiant du maitre*/
int slaves[num_slaves]; /*tableau des id des esclaves*/
int **a,**b,**c;
int tab_ind[num_elm]; /*tableau des indices i j*/
int result;
int results[num_send];
a=malloc(dim*sizeof(int*));
if(a==NULL)
{
fprintf(stderr,"Allocation impossible");
exit(EXIT_FAILURE);
}
for(i=0;i<dim;i++)
{
a[i]=calloc(dim,sizeof(int));
if(a[i]==NULL)
{
fprintf(stderr,"Allocation impossible");
exit(EXIT_FAILURE);
}
}
b=malloc(dim*sizeof(int*));
if(b==NULL)
{
fprintf(stderr,"Allocation impossible");
exit(EXIT_FAILURE);
}
for(i=0;i<dim;i++)
{
```

```
b[i]=calloc(dim,sizeof(int));
if(b[i]==NULL)
{
fprintf(stderr,"Allocation impossible");
exit(EXIT_FAILURE);
}
}
c=malloc(dim*sizeof(int*));
if(c==NULL)
{
fprintf(stderr,"Allocation impossible");
exit(EXIT_FAILURE);
}
for(i=0;i<dim;i++)
{
c[i]=calloc(dim,sizeof(int));
if(c[i]==NULL)
{
fprintf(stderr,"Allocation impossible");
exit(EXIT_FAILURE);
}
}
/*definir le tableau des indices*/
for(i=0;i<num_elm;i++)
tab_ind[i]=i+1;
for(i=0;i<dim;i++)
{
for(j=0;j<dim;j++)
{
a[i][j]=i+j;
c[i][j]=0;
if(i==j)
b[i][j]=1;
else
b[i][j]=0;
}
}
/*declaration du maitre et obtention de l'identifiant*/
mytid=pvm_mytid();
result=pvm_spawn("slave_multi_mat",(char**)0,PvmTaskDefault,"",num_slaves,slaves);
/*verification du lancement de l'esclave*/
if(result != num_slaves)
{
```

```
fprintf(stderr,"Error : Cannot spawn slave.n");
/*sortie de pvm*/
pvm_exit();
exit(EXIT_FAILURE);
}
/*envoi des indices*/
for(i=0;i<num_slaves;i++)
{
/*initialisation des donnees a envoyer a l'esclave*/
pvm_initsend(PvmDataDefault);
/*insertion des indices dans le buffer*/
pvm_pkint(tab_ind+i*num_send,num_send,1);
/*envoi des indices*/
pvm_send(slaves[i],msg_ind);
}
/*reception des resultats*/
for(k=0;k<num_slaves;k++)
{
int bufid, bytes,type,source,slave_no;
/*reception des resultat d'un esclave quelquonque*/
bufid=pvm_recv(-1,msg_result);
/*obtenir les information sur le message recu (l'id de l'esclave, la taille...)*/
pvm_bufinfo(bufid,&bytes,&type,&source);
/*determiner l'esclave qui a envoyer le resultat*/
slave_no=get_slave_no(slaves,source);
/*stockage des resultats*/
pvm_upkint(results,num_send,1);
/*decodage des indices et recuperation de elements c[i][j]*/
for(l=0;l<num_send;l++)
{
j=tab_ind[slave_no*num_send+l]%dim;
if(j==0)
j=dim;
i=(tab_ind[slave_no*num_send+l]-j)/dim;
j+=-1;
c[i][j]=results[l];
}
}
/*enregister le resulat dans un fichier*/
FILE *f;
f=fopen("result_multi_mat_pvm.txt", "w");
if (f!= NULL)
{
```

```
fprintf(f,"a=\n");
for(i=0;i<dim;i++)
{
fprintf(f,"\n");
for(j=0;j<dim;j++)
fprintf(f,"%d ,", a[i][j]);
fprintf(f,";");
}
fprintf(f,"\n");
fprintf(f,"b=\n");
for(i=0;i<dim;i++)
{
fprintf(f,"\n");
for(j=0;j<dim;j++)
fprintf(f,"%d ,", b[i][j]);
fprintf(f,";");
}
fprintf(f,"\n");
fprintf(f,"c=\n");
for(i=0;i<dim;i++)
{
fprintf(f,"\n");
for(j=0;j<dim;j++)
fprintf(f,"%d ,", c[i][j]);
fprintf(f,";");
}
fprintf(f,"\n");
fclose(f);
}
else
perror("result_multi_mat_pvm.txt");
fin=clock(); /*fin de la mesure de la duree d'execution*/
/*reception de la dure d'execution des esclave*/
for(i=0;i<num_slaves;i++)
{
int bufid, bytes,type,source,slave_no;
/*reception des resultat d'un esclave quelquonque*/
bufid=pvm_recv(-1,msg_time);
/*obtenir les information sur le message recu (l'id de l'esclave, la taille...)*/
pvm_bufinfo(bufid,&bytes,&type,&source);
/*determiner l'esclave qui a envoyer le resultat*/
slave_no=get_slave_no(slaves,source);
/*stockage des resultats*/
```

```
pvm_upkfloat(time+slave_no,1,1);
}
ts=0;
for(i=0;i<num_slaves;i++)
{
if(ts<time[i]) ts=time[i];
}
temps=(fin-debut)/clk_sec + ts;
for(i=0;i<num_slaves;i++)
printf("duree d'execution de l'esclave %d = %f\n",(i+1),time[i]);
printf("temps d'execution totale = %f secondes \n",temps);
/*sortie de pvm*/
pvm_exit();
exit(EXIT_SUCCESS);
return(0);
}
/*fonction qui retourne le numero de l'esclave*/
int get_slave_no(int *slaves,int task_id)
{
int i;
for(i=0;i<num_slaves;i++)
{
if(slaves[i]==task_id) return i;
}
return -1;
}
```

```
programme esclave*/

#include<stdio.h>

#include<stdlib.h>

#include<time.h>

#include"pvm3.h"

#include"tags.h"

#include"common.h"

int main()

{

float ts;

float clk_sec=CLOCKS_PER_SEC;
```

```
clock_t debut,fin;
debut=clock(); /*debut de la mesure de la duree d'execution*/
int **a,**b;
int tab_ind[num_send];
int i,j,k,l;
int mytid;
int parent_tid;
a=malloc(dim*sizeof(int*));
if(a==NULL)
{
fprintf(stderr,"Allocation impossible");
exit(EXIT_FAILURE);
}
for(i=0;i<dim;i++)
{
a[i]=calloc(dim,sizeof(int));
if(a[i]==NULL)
{
fprintf(stderr,"Allocation impossible");
exit(EXIT_FAILURE);
}
}
      b=malloc(dim*sizeof(int*));
      if(b==NULL)
{
fprintf(stderr,"Allocation impossible");
exit(EXIT_FAILURE);
}
      for(i=0;i<dim;i++)
{
b[i]=calloc(dim,sizeof(int));
if(b[i]==NULL)
{
```

```
fprintf(stderr,"Allocation impossible");

exit(EXIT_FAILURE);

}

}

/*matrice a , b *
/
for(i=0;i<dim;i++)

{

for(j=0;j<dim;j++)

{

a[i][j]=i+j;

if(i==j)

b[i][j]=1;

else

b[i][j]=0;

}

}

    /*prendre id*
/
mytid=pvm_mytid();

/*obtention de l'identifiant du maitre*
/
parent_tid=pvm_parent();

/*reception des donees*
/
pvm_recv(parent_tid,msg_ind);

pvm_upkint(tab_ind,num_send,1);

/*decodage des indices i et j et calcul de c[i][j]*
/
for(k=0;k<num_send;k++)

{

j=tab_ind[k]%dim;

if(j==0)j=dim;

i=(tab_ind[k]-j)/dim;

j+=-1;

tab_ind[k]=0;

for(l=0;l<dim;l++)
```

```
tab_ind[k]+=a[i][l]*b[l][j];

}

/*emission du resultat*
/
pvm_initsend(PvmDataDefault);

pvm_pkint(tab_ind,num_send,1);

pvm_send(parent_tid,msg_result);

fin=clock(); /*fin de la mesure de la duree d'execution*
/
ts=0;

ts=(fin-debut)/clk_sec; /*clacule de la duree d'execution en secondes*
/
/*envoi de la duree d'execution au maitre*
/
pvm_initsend(PvmDataDefault);

pvm_pkfloat(&ts,1,1);

pvm_send(parent_tid,msg_time);

/*sortie de pvm*
/
pvm_exit();

exit(EXIT_SUCCESS);

return(0);

}
```

```
#makefile

#chemain vers les bibliotheque

INCDIR=-I$(HOME)/pvm3/include

LIBDIR=-L$(HOME)/pvm3/lib/$(PVM_ARCH)

LIBS=-lpvm3

CFLAGS=-Wall

CC=gcc

TARGET=all

PVM_HOME=$(HOME)/pvm3/bin/LINUX

all:$(PVM_HOME)/master_multi_mat $(PVM_HOME)/slave_multi_mat

$(PVM_HOME)/master_multi_mat: master_multi_mat.c common.h tags.h

    $(CC)  -o $(PVM_HOME)/master_multi_mat master_multi_mat.c $(CFLAGS) $(LIBS) $(INCDIR) $(LIBDIR)

$(PVM_HOME)/slave_multi_mat: slave_multi_mat.c common.h tags.h
```

```
    $(CC)  -o $(PVM_HOME)/slave_multi_mat slave_multi_mat.c $(CFLAGS) $(LIBS) $(INCDIR) $(LIBDIR)
clean:
    rm $(HOME)/pvm3/bin/LINUX/slave_multi_mat
    rm $(HOME)/pvm3/bin/LINUX/master_multi_mat
    clear
```

D.2 Stress test

```
    /*stress test*/
/* compiler avec gcc -Wall stress.c -o stress*/
#include <stdio.h>
#include<stdlib.h>
#include<time.h>
#define dim 1000
int main (void)
{
clock_t debut,fin;
float temps;
debut=clock();
int **a,**b,**c;
int i,j,k;
a=malloc(dim*sizeof(int*));
if(a==NULL)
{
fprintf(stderr,"Allocation impossible");
exit(EXIT_FAILURE);
}
for(i=0;i<dim;i++)
{
a[i]=calloc(dim,sizeof(int));
if(a[i]==NULL)
{
fprintf(stderr,"Allocation impossible");
exit(EXIT_FAILURE);
}
}
b=malloc(dim*sizeof(int*));
if(b==NULL)
{
```

```
fprintf(stderr,"Allocation impossible");
exit(EXIT_FAILURE);
}
for(i=0;i<dim;i++)
{
b[i]=calloc(dim,sizeof(int));
if(b[i]==NULL)
{
fprintf(stderr,"Allocation impossible");
exit(EXIT_FAILURE);
}
}
c=malloc(dim*sizeof(int*));
if(c==NULL)
{
fprintf(stderr,"Allocation impossible");
exit(EXIT_FAILURE);
}
for(i=0;i<dim;i++)
{
c[i]=calloc(dim,sizeof(int));
if(c[i]==NULL)
{
fprintf(stderr,"Allocation impossible");
exit(EXIT_FAILURE);
}
}
for(i=0;i<dim;i++)
{
for(j=0;j<dim;j++)
{
a[i][j]=i+j;
c[i][j]=0;
if(i==j)
b[i][j]=1;
else
b[i][j]=0;
}
}
for(i=0;i<dim;i++)
{
for(j=0;j<dim;j++)
{
```

```
for(k=0;k<dim;k++)
c[i][j]+=a[i][k]*b[k][j];
}
}
fin=clock();
temps=(fin-debut)/CLOCKS_PER_SEC;
printf("temps d'execution = %f \n",temps);
return(1);
}
```

```
#!/bin/bash
#test associe a stress test
i=1
echo > resultat
while [ $i -lt 4 ]
do
./stress >> resultat&
echo $i
i=‘expr $i + 1‘
done
```

Lexique

CPU : Central Processing Unit
DFSA : Direct File System Access
DHCP : Dynamic Host Configuration Protocol
DNS : Domain Name System
FTP : File Transfer Protocol
HA : High Availibility
HPC : High Performance Computing
MFS : Mosix File System
NFS : Network File System
PPM : Preemptive Process Migration
PVM : Parallel Virtual Machine
PVP : Parallel Vectorial Processor
PXE : Preboot Execution Environment
RAID : Redundant Array of Inexpensive Disks
SMP : Symetric Multi-Processor
SSH : Secure Shell
TCP : Transmission Control Protocol
UDP : Universal Datagram Protocol
UHN : User Home Node
XDR : eXternal Data Representation

Bibliographie

[1] www.top500.org.

[2] Franck CAPPELLO et Jean-Paul SANSONNET. *Introduction au parallélisme et aux architectures parallèles.* Techniques de l'Ingénieur.

[3] Dictionnaire Larousse.

[4] Carl Kesselman. *The Grid Blueprint for a new Computing Infrastructure.* ELSVIER, 1998.

[5] Matt Rechenburg. *openmosix.* man page.

[6] Dr. Moshe Bar et MAASK. http ://openmosix.sourceforge.net/linux-kongress_2003_openMosix.pdf, 2003.

[7] Maurice Libes. *Utilisation d'un cluster de calcul openMOSIX et déploiement à l'aide de LTSP.* Université de la Méditerranée. Marseille, 2003.

[8] Garandeau Edouard et Steenhoute Vincent. *OpenMosix.* Ecole Supérieur d'Informatique de Paris, 2004.

[9] http ://www.openmosixview.com/.

[10] http ://www.emse.fr/ ~corbel/PARALLEL/pvm.html.

[11] *OpenMosix on Debian Sarge.* http ://cluster-lab.bzzz.net/doku.php ?id=openmosixondebiansarge, 2007.

المُلَخَصْ

الكلُوسترِنغ هُوَ عِبَارَة عَن مَجمُوعَة مِنَ الطُرُق الَتِي تَسمَح بِإنجَاز المَهَام المُتَوَازِيَة مِن اجلِ تَحسِين فَعَالِيَة وَ جَوْدَة النِظَام.
هَـذَا العَمَل يَتَمَثَل فِي دِرَاسَة طُرُق الكلُوسترِنغ ذَات المَصَادِر المَفتُوحَة: OpenMosix وَ PVM. OpenMosix يَسمَح بِتَوزِيع الحُمُولَة وَ تَقسِيمِهَا علَى كَامِل النِظَام أَيْ عَلَى كَامِل العُقَدْ، وَ مِن نَاحِيَة أُخرَى PVM هُوَ عِبَارَة عَن بَرنَامَج يَسمَح بِجَمع شَبَكَة مِن أَجهِزَة LINUX لَا مُتَجَانِسَة لِتُصبِح ظَاهِرَة عَلَى شَكل اجهِزَة مُتَوَازِيَة.
الكَلِمَات المِفتَاحِيَة : الكلُوسترِنغ ,كلُوستَر ,المَصَادِر المَفتُوحَة ,LINUX, PVM, OpenMosix.

Résumé

Le clustering, se définit comme l'opération de mettre un ensemble de machines et de l'organiser de telle façon à ce qu'elle travaille en parallèle et exécute un nombre important de tâches. Ces techniques de clustering permettent de gagner en performance. Notre travail s'inscrit dans la mise en œuvre d'un cluster en utilisant deux solutions open source sous Linux : OpenMosix et PVM. OpenMosix permet d'équilibrer la charge dans le cluster tandis que PVM permet d'exhiber le parallélisme d'une application.
Mots clés : Clustering, cluster, parallélisme, solution libre, OpenMosix, PVM, Linux.

Abstract

The clustering is an ensemble of methods that enables parallel tasks in order to improve the efficiency and the performance of the system. Our work consists of the study of open source clustering methods : OpenMosix and PVM. OpenMosix clusters enable load-balancing and divides the load of the whole system on the different nodes. On the other hand, PVM is a software system that permits heterogeneous collection of Linux computers networked to be viewed as a parallel computer.
Key words : Clustering, cluster, open source, OpenMosix, PVM, Linux, parallel.