Systèmes à image unique pour grappes : Une étude comparative

Benoit Boissinot
Benoit.Boissinot@ens-lyon.fr
E.N.S Lyon

sous la direction de

Christine Morin
Christine.Morin@irisa.fr
IRISA

Renaud Lottiaux
Renaud.Lottiaux@irisa.fr
IRISA

6 septembre 2004

Table des matières

1 Introduction

Ce stage effectué au sein du projet INRIA/PARIS à l'IRISA s'inscrit dans le contexte de l'activité de recherche Kerrighed. Cette activité de recherche vise à développer un système d'exploitation pour les grappes de calculateurs.

Dans le monde du calcul haute performance, les grappes de calculateurs proposent une architecture évolutive et qui ont un meilleur rapport performance/coût que les supercalculateurs traditionnels.

Dans le but de fournir une interface globale au dessus d'une grappe, les systèmes à image unique ont été développés. L'objectif d'un système à image unique pour grappe est de donner l'illusion d'une machine unique à l'administrateur, au programmeur et à l'utilisateur en proposant un accès global aux ressources processeurs, mémoire et disque.

Nous avons effectué une étude comparative des fonctionnalités et des performances de différents systèmes à image unique. Nous avons choisi d'évaluer les systèmes OpenMosix, OpenSSI et Kerrighed, tout trois ayant comme système hôte Linux et le code source disponible. Nous avons installé la dernière version de ces systèmes sur une grappe de l'IRISA et nous avons exécuté un ensemble de tests sur tous les systèmes.

Dans la première partie de ce document, nous présentons les trois systèmes. Ensuite nous comparons leurs fonctionnalités et enfin nous présentons les résultats de l'évaluation des performances. Nous dressons un bilan de notre étude en conclusion.

2 Systèmes à image unique

2.1 Généralités

Définition Un système à image unique est un dispositif logiciel ou matériel créant l'illusion qu'un ensemble de ressources (logiques ou physiques) distribuées ne forment qu'une ressource unique et partagée.

Un système à image unique permet donc de rendre transparent l'accès aux ressources d'une grappe, dans le but de mieux les exploiter.

Dans le suite de ce paragraphe, nous présentons brièvement les trois systèmes étudiés et nous mentionnons les autres systèmes de ce type décrits dans la littérature.

2.2 Mosix / OpenMosix

Le développement de Mosix[2] a commencé en 1981, à l'université de Jérusalem, le système s'appelait à l'époque MOS (Multi-computer Operating System) et tournait sur un système hôte de type BSD.

En 1999, Mosix a été porté sur le système Linux. En 2002, suite à des divergences concernant le futur de Mosix, OpenMosix[1] a été créé, à partir de la base de code de Mosix.

L'objectif initial de Mosix était de proposer un ordonnancement global des processus dans la grappe en faisant migrer des processus pour équilibrer la charge. D'autres composants permettant de globaliser les ressources ont été rajoutés par la suite, comme par exemple le système de fichiers global. La volonté actuelle d'OpenMosix est d'être un véritable système à image unique, notamment

avec comme objectif une mémoire partagée distribuée afin de permettre la migration de threads ou de processus utilisant des segments de mémoire partagée.

2.3 OpenSSI

OpenSSI[14] est un système dont la base de code provient du logiciel NonStop Cluster[10]. Ce logiciel développé par HP proposait une solution pour construire une grappe avec le système UnixWare.

En 2001, le projet OpenSSI est formé afin de proposer une solution équivalente sous Linux. En plus du code provenant initialement de NonStop Cluster, HP intègre d'autres projets du monde Linux (Distributed Lock Manager[15], haute disponibilité[16], Linux Virtual Server[17],...).

2.4 Kerrighed

Le développement du système Kerrighed[5][11], a été commencé en 1999, à l'IRISA (Rennes, France). Ce système propose une image unique grâce à l'utilisation de concepts génériques :

- Processus fantômes : utilisés pour la migration des processus et les points de reprise.
- Conteneurs : utilisés pour la mémoire partagée distribuée et le cache global.
- Flux dynamiques : utilisés pour la migration des sockets et des tubes.[3]

2.5 Autres systèmes

Il existe d'autres systèmes à image unique, presque tous développés dans un cadre universitaire. Parmi les plus représentatifs, nous pouvons citer :

- Amoeba[9], ce système a été écrit de zéro à l'université de Vrije (Pays-Bas). Le développement a commencé en 1981. Actuellement ce système n'est plus maintenu.
- Sprite[6], également écrit de zéro à l'université de Californie (Berkeley, USA), à peu près à la même époque que Amoeba. Sprite n'est également plus maintenu.
- Plurix[8], écrit de zéro à l'université d'Ulm (Allemagne). Le développement a commencé en 1993 et le système est toujours actif. Ce système utilise un concept de transaction associé à une mémoire partagée distribuée.
- Genesis[4], écrit de zéro à l'université Deakin (Australie) dans les années 1995. Ce système est fondé sur les technologies de type micro-noyau. Un portage sur le système Linux est en cours.
- Nomad[7], fondé sur Linux, conçu à l'université fédérale de Rio de Janeiro (Brésil) à partir de 1996. Le projet n'est actuellement plus actif.
- DragonflyBSD[13], fondé sur FreeBSD-4.x, dont le développement a commencé en 2003. C'est un projet récent d'un groupe de développeurs système BSD et le système à image unique est pour eux un but à long terme (quelques années).

3 Étude des fonctionnalités

Parmi les systèmes décrits dans la littérature, nous avons retenu les trois systèmes fondés sur Linux dont le code est distribué en Open Source pour une étude expérimentale. Ces trois systèmes sont OpenMosix, OpenSSI et Kerrighed.

3.1 Installation des systèmes

OpenMosix L'installation d'OpenMosix sous Debian GNU/Linux s'effectue en deux étapes. Il faut d'abord compiler puis installer un noyau préalablement patché. Ensuite, à l'aide des outils de gestion de paquetages, on installe les logiciels et outils spécifiques à OpenMosix. L'installation s'effectue sans difficulté particulière. Il y a peu de fichiers de configuration à modifier et la documentation est claire.

OpenSSI L'installation des outils utilise le système de gestion des paquetages, et ne pose donc pas de problème. En revanche, la mise en place des nœuds peut poser des difficultés lorsque la configuration de la grappe s'éloigne du modèle de la documentation (notamment une installation dans un réseau non privé).

Kerrighed L'installation se fait à partir d'une archive et en suivant les instructions des notes d'installation (utilisant les outils standards d'installation GNU/Linux). Malgré le peu d'intégration dans la distribution, la configuration se fait de manière simple et le processus est détaillé.

3.2 Vue globale de la grappe

Sur un système Linux standard, le système d'exploitation exporte des données dans le répertoire `/proc`. Ces informations (charge, processus actifs, ...) sont utilisées par des outils comme `ps` ou `top`. Nous avons d'abord vérifié l'existence d'outils donnant des informations équivalentes sur l'ensemble de la grappe (soit avec un export par `/proc`, soit avec un outil spécifique). Nous avons également testé l'existence d'une méthode permettant d'accéder à l'ensemble des fichiers de la grappe (système de fichiers global) ou permettant d'accéder à des périphériques distants.

Le tableau 1 propose une synthèse des résultats obtenus.

OpenMosix Les outils `mtop, mps` permettent d'obtenir des informations sur les processus dans la grappe. Ces utilitaires ne sont pas globaux dans le sens où ils n'affichent pas tous les processus de la grappe. Par exemple si `mtop` est lancé sur le nœud A, parmi les processus distants, seuls ceux ayant été lancés sur le nœud A puis migrés seront visibles. Le logiciel *openmosixview* offre une interface graphique pour surveiller la charge et l'utilisation mémoire des différents nœuds.

Le système de fichiers oMFS (OpenMosix File System) permet d'accéder à la racine de tous les nœuds par le chemin `/mfs/`*`num_noeud`*`/`.

Les périphériques sont propres à chaque nœud : un nœud ne peut pas accéder aux périphériques d'un autre nœud.

OpenSSI Les utilitaires non modifiés `top`, `ps` permettent d'obtenir des informations sur l'ensemble des processus s'exécutant sur la grappe (les PID sont globalisés). En revanche les données concernant la mémoire ou la charge ne sont pas globalisées et sont locales au nœud.

Le système de fichiers est global à toute la grappe. En effet la racine du nœud d'*init* est partagée avec les autres nœuds. Si l'on monte une partition sur un nœud quelconque on peut choisir si le montage est global ou local.

Les périphériques sont également accessibles depuis n'importe quel nœud, par le répertoire `/dev/node#/`.

Kerrighed Les outils non modifiés `top`, `ps` donnent des informations sur l'ensemble de la grappe pour ce qui est de la mémoire et de l'utilisation CPU, on peut voir la grappe comme une unique machine SMP. En revanche seul les processus s'exécutant localement sont visibles avec ces outils.

Le système de fichiers ContainerFS permet de partager l'ensemble des fichiers de la grappe à travers un unique point de montage.

De la même facon que OpenMosix, les périphériques sont propres à chaque nœud. De plus un processus après migration ne conserve pas la console de son nœud d'origine, à l'inverse des deux autres systèmes.

Vue globalisée	OpenMosix	OpenSSI	Kerrighed
Système de fichiers	oui	oui	oui
Processus (`ps`)	non[1]	oui	non
Charge mémoire et processeur (`top`)	non[1]	non	oui
Périphériques (`/dev`)	non	oui	non

TAB. 1 – Ressources globalisée du point de vue de l'administrateur/utilisateur.

3.3 Ordonnancement des tâches

Les trois systèmes proposent un mécanisme d'ordonnancement global. Les processus s'exécutant sur des nœuds déjà chargés sont déplacés vers un nœud ayant plus de ressources disponibles. OpenMosix et OpenSSI proposent le même algorithme d'ordonnancement, l'ordonnanceur de Kerrighed est configurable à chaud. Il peut être configuré de façon à avoir le même comportement que les deux autres systèmes.

3.4 Objets systèmes migrables

Nous avons testé la migration de différents objets systèmes : tubes et sockets, sémaphores, mémoire partagée, threads d'un processus. Le tableau 2 présente une synthèse des fonctionnalités des différents systèmes.

OpenMosix Les processus séquentiels peuvent migrer avec OpenMosix.

En ce qui concerne les communications inter-processus, les mécanismes suivant sont supportés et n'empêchent pas la migration : socket INET/UNIX, sémaphores Sys V, tubes.

Les applications multithreadées ou utilisant de la mémoire partagée ne peuvent pas migrer.

OpenSSI OpenSSI permet de faire migrer des processus séquentiels.

Les socket INET/UNIX, les sémaphores Sys V, les tubes ainsi que la mémoire partagée Sys V peuvent être utilisés par des processus sans que cela empêche la migration.

La mise en œuvre actuelle du partage de mémoire entre processus dans OpenSSI n'est pas utilisable. Par exemple l'algorithme modifié de Gram-Schmidt (MGS) avec une matrice 16x16 et deux processus s'exécute en moins de 1/10ème de secondes si les deux processus se situent sur le

[1] Informations disponibles par un logiciel spécifique, et avec des restrictions.

même nœud tandis que le calcul dure 47 minutes lorsque les deux processus sont sur des nœuds différents.

Les applications multithreadées ne peuvent pas migrer.

Kerrighed Les processus séquentiels peuvent migrer avec Kerrighed.

L'utilisation des mécanismes de communication suivants n'empêche pas les migrations de processus : socket INET/UNIX, mémoire partagée Sys V.

Les tubes ne sont pas encore mis en œuvre dans Kerrighed, de même que les sémaphores Sys V (les sémaphores POSIX sont présents).

Kerrighed permet de faire migrer des applications multithreadées ainsi que des threads individuels.

Structures migrables	OpenMosix	OpenSSI	Kerrighed
Segment mémoire Sys V	non	oui	oui
Sémaphores Sys V	oui	oui	non
Socket UNIX/INET	oui	oui	oui
Tubes	oui	oui	non
Thread	non	non	oui

TAB. 2 – Objets systèmes pouvant être migrés.

3.5 Tolérance aux fautes et haute disponibilité

Dans le cadre de la tolérance aux fautes et de la haute disponibilité, nous avons testé les points suivants : arrêt brutal d'un nœud, création de points de reprise, ajout ou retrait d'un nœud à chaud. Le tableau 3 présente une synthèse pour les différents systèmes.

OpenMosix Pour ajouter ou enlever un nœud à une grappe OpenMosix, il suffit de lancer ou d'arrêter le service `/etc/init.d/openmosix`. En cas d'arrêt brutal d'un nœud, la grappe continue de tourner, les processus tournant sur le nœud défaillant étant perdus.

OpenSSI Dans OpenSSI, un nœud s'ajoute à la grappe lorsqu'il démarre, les nœuds peuvent être rajoutés à chaud. De plus la primitive `clusternode_shutdown` permet de retirer un nœud de la grappe. Un nœud ne peut revenir dans la grappe qu'après un redémarrage du nœud. OpenSSI supporte l'arrêt brutal d'un nœud à condition que ce ne soit pas le nœud hébergeant la racine du système de fichiers.

Kerrighed Les nœuds ne peuvent pas être ajoutés ou retirés à chaud, la grappe ne peut démarrer que lorsque tous les nœuds spécifés dans le fichier de configuration sont en marche. Kerrighed ne supporte pas le retrait brutal d'un nœud. En revanche il y a une possibilité de faire des points de reprise sur un processus séquentiel pour sauvegarder son état courant et le relancer plus tard.

	OpenMosix	OpenSSI	Kerrighed
Ajout de nœuds à chaud	oui	oui	non
Retrait de nœuds à chaud	oui	oui	non
Tolérance aux la pannes	oui	oui	non
Point de reprise	non	non	oui[2]

TAB. 3 – Tolérance aux fautes et haute disponibilité.

3.6 Support matériel

Les processeurs 64 bits sont supportés par OpenMosix (Itanium) et OpenSSI (Alpha, Itanium). De plus ces deux systèmes fonctionnent sur des machines multi-processeurs (SMP). Kerrighed ne fonctionne sur aucune architecture 64 bits, et ne supporte pas les systèmes multi-processeurs.

	OpenMosix	OpenSSI	Kerrighed
Support 64 bits	oui	oui	non
Support multi-processeurs	oui	oui	non

TAB. 4 – Architectures supportées.

4 Étude des performances

4.1 Méthodologie

Le système utilisé est une grappe de 4 nœuds utilisant la distribution Debian GNU/Linux (Sid) avec les caractéristiques matérielles suivantes :

- Processeur Intel Pentium III (1 GHz)
- Connexion Fast Ethernet (carte Intel eepro100)
- Disques IDE

Nous avons testé OpenMosix-2.4.22-3 fondé sur le noyau Linux 2.4.22, OpenSSI-1.0.0-rc5 fondé sur le noyau 2.4.20 et Kerrighed-1.0-rc4 fondé sur un noyau 2.4.24.

Nous avons testé l'impact de la migration sur l'exécution d'un processus, mesuré l'effet de la migration sur les performances des communications inter-processus et testé le système de fichiers.

Pour effectuer les tests sur la migration, nous avons utilisé des programmes séquentiels simples (somme de vecteurs).

Les tests de communication ont été effectués avec le logiciel NetPipe (version 3.6.1). Nous en avons modifié les sources pour effectuer la migration et pour ajouter le support des sockets UNIX et des tubes.

Les tests sur le système de fichiers ont été effectués à l'aide de simples programmes lisant ou écrivant des fichiers.

Les tests classiques de benchmarks SMP n'ont pas pu être effectués, car certains systèmes ne proposent pas de mécanisme permettant de partager la mémoire (threads ou mémoire partagée

[2] Uniquement pour les processus séquentiels.

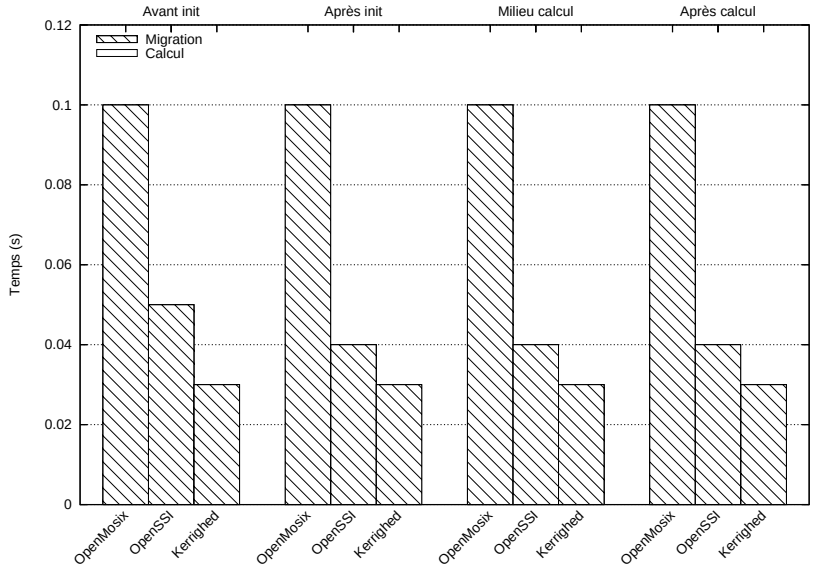

FIG. 1 – Migration vs. calcul, taille des données 16 Ko.

System V). Nous avons également voulu tester la compilation parallèle du noyau, mais l'absence de migration de tubes avec Kerrighed et les erreurs rencontrées avec OpenMosix, ne nous ont pas permis faire ce test.

4.2 Migration des processus

Nous avons mesuré le temps de migration des processus en mesurant le temps d'exécution d'une application séquentielle simple (somme de vecteurs) avec et sans migration. Les mesures ont été efféctuées avec une unique migration de l'application à différents moments de son exécution : avant initialisation des vecteurs, après initalisation, au milieu du calcul et après le calcul.

Les figures 1 et 2 montrent les résultats obtenus avec des vecteurs de tailles respectives 16 Ko et 64 Mo.

Dans le premier cas, le temps de calcul est négligeable par rapport au temps de migration, de plus les valeurs étant petites et l'outil de mesure (`/usr/bin/time`) relativement peu précis, la seule conclusion que l'on peut tirer est que OpenMosix est légèrement plus lent que les deux autres. Mais cela peut être dû à une transmission des signaux moins rapide que dans les autres systèmes.

Lorsqu'on regarde les résultats obtenus avec des vecteurs de taille 64 Mo, on observe que OpenMosix obtient les meilleurs résultats, suivi par Kerrighed puis par OpenSSI lorsque la migration s'effectue après l'initialisation. Le test où la migration est effectuée au milieu ou après le calcul met en valeur les différents choix de mise en œuvre. Alors que OpenMosix et OpenSSI migrent systématiquement tout l'espace d'adressage du processus, Kerrighed migre les pages à la demande et la migration est donc plus rapide.

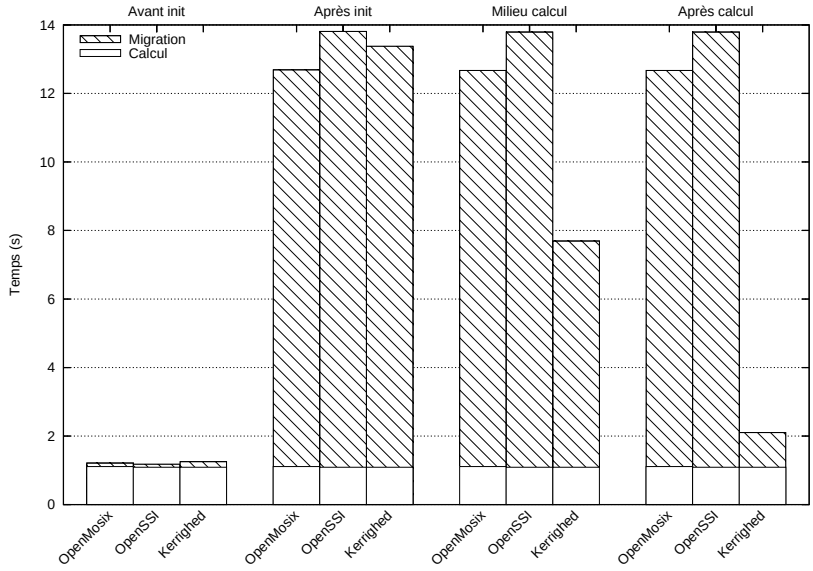

FIG. 2 – Migration vs. calcul, taille des données 64 Mo.

4.3 Communication inter-processus

A l'aide du logiciel NetPipe, qui permet de tester les performances des communications entre processus, nous avons testé les performances des principaux moyens de communications inter-processus (socket INET/UNIX, tube).

Les tests ont pour but de mesurer la pénalité induite par la migration dans les communications (latence et bande passante). À cet effet, les tests suivants ont été effectués :

- Test 1 (INET), figures 3 et 5 :
  - Création des extrémités : extrémité A sur le nœud 1 et extrémité B sur le nœud 2.
  - Migration de l'extrémité B sur le nœud 3.
- Test 2 (INET), figures 4 et 5 :
  - Création des extrémités : extrémité A sur le nœud 1 et extrémité B sur le nœud 2.
  - Migration de l'extrémité A sur le nœud 3, migration de l'extrémité B sur le nœud 4.
- Test 3 (INET), figure 6 :
  - Création des extrémités : extrémités A et B sur le nœud 1.
  - Migration de l'extrémité A sur le nœud 2, puis migration de l'extrémité B également sur le nœud 2.
- Test 4 (UNIX), figure 7 :
  - Création des extrémités : extrémités A et B sur le nœud 1.
  - Migration des extrémités A et B sur le nœud 2.
- Test 5 (UNIX), figure 8 :
  - Création des extrémités : extrémités A et B sur le nœud 1.
  - Migration de l'extrémité A sur le nœud 2, puis migration de l'extrémité B sur le nœud 3.

- Test 6 (tubes), figure 9 :
  - Création des extrémités : extrémités A et B sur le nœud 1.
  - Migration des extrémités A et B sur le nœud 2.
- Test 7 (tubes), figure 10 :
  - Création des extrémités : extrémités A et B sur le nœud 1.
  - Migration de l'extrémité A sur le nœud 2, puis migration de l'extrémité B sur le nœud 3.

Les tests ayant pour objets les sockets INET sont les seuls ayant été satisfaisants sur les trois systèmes (des problèmes de stabilité ont gêné la prise de mesure sur Kerrighed).

Sur ces tests, dans le cas de la migration d'une extrémité, on observe une diminution importante des performances pour OpenMosix (latence : +154%, bande passante maximale : −42%) et OpenSSI (latence : +403%, bande passante maximale : −74%). Les performances se détériorant encore plus après avoir migré les deux extrémités.

Au contraire, les performances restent identiques après migration pour le système Kerrighed.

Ces résultats sont principalement dûs au fait que contrairement à OpenSSI et OpenMosix qui mettent en place un système de relais, Kerrighed possède une gestion des flux dynamiques.[3]

Les tests sur les sockets UNIX et les tubes n'ont pu être effectués que sur les systèmes OpenMosix et OpenSSI. En effet les tubes ne sont pas encore migrables dans Kerrighed, et l'utilisation des sockets UNIX est encore instable.

Les résultats de ces tests montrent encore que OpenMosix et OpenSSI utilisent un système de relais. On peut voir sur les figures 7, 8, 9 et 10 un affaiblissement de la bande passante si l'on migre les deux extrémités. De même que pour les sockets TCP, OpenMosix reste plus performant que OpenSSI dans tous les tests. On peut aussi remarquer que lors de la migration des deux extrémités sur le même nœud, on obtient un débit très éloigné du débit que l'on pourrait attendre (débit mémoire), il s'agit également d'un inconvénient du système de relais.

4.4 Système de fichiers global

Les différents tests effectués sur le système de fichiers avaient pour but de tester les performances en lecture et en écriture en vérifiant l'existence et les performances d'un cache global. Nous avons donc effectué les tests suivants :

- Lecture (figure 11)
  1. lecture à froid d'un fichier sur le nœud où il se trouve physiquement (débit attendu : débit du disque)
  2. relecture (débit attendu : débit de la mémoire)
  3. lecture du même fichier sur un autre nœud (débit attendu : débit du réseau)
  4. relecture (débit attendu : débit de la mémoire)
- Lecture avec migration (figure 12)
  1. lecture à froid d'un fichier sur le nœud où il se trouve physiquement (débit attendu : débit du disque)
  2. relecture (débit attendu : débit de la mémoire)
  3. migration sur un autre nœud puis lecture du même fichier (débit attendu : débit du réseau)
  4. relecture (débit attendu : débit de la mémoire)

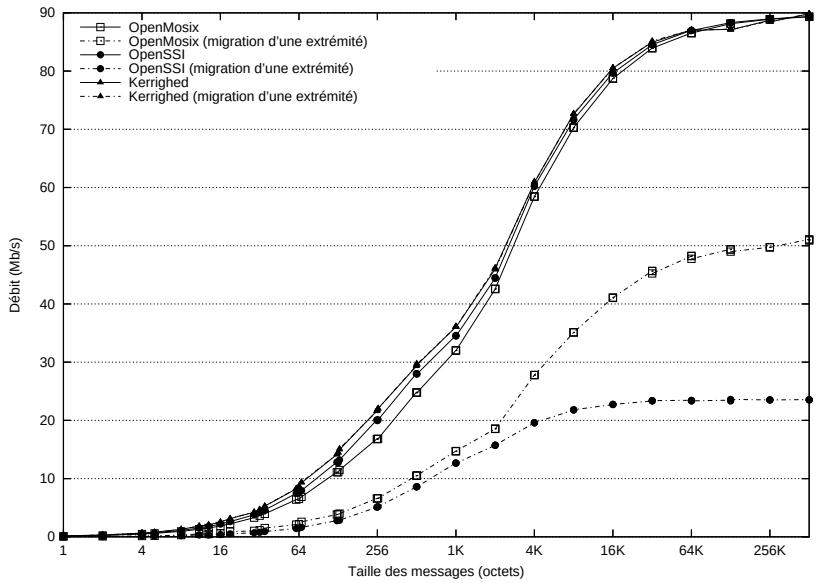

FIG. 3 – Bande passante des communications par socket TCP, création des extrémités sur les nœuds 1 et 2 puis migration d'une extrémité.

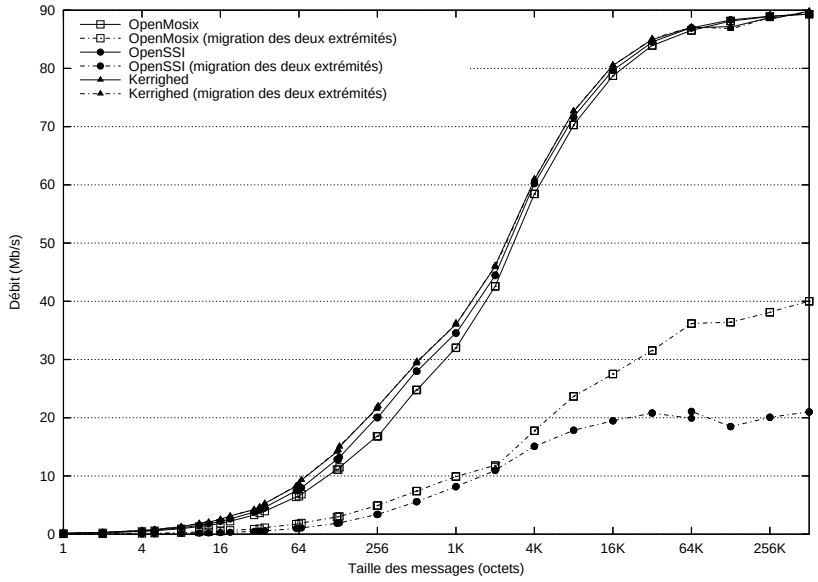

FIG. 4 – Bande passante des communications par socket TCP, création des extrémités sur les nœuds 1 et 2 puis migration des deux extrémités.

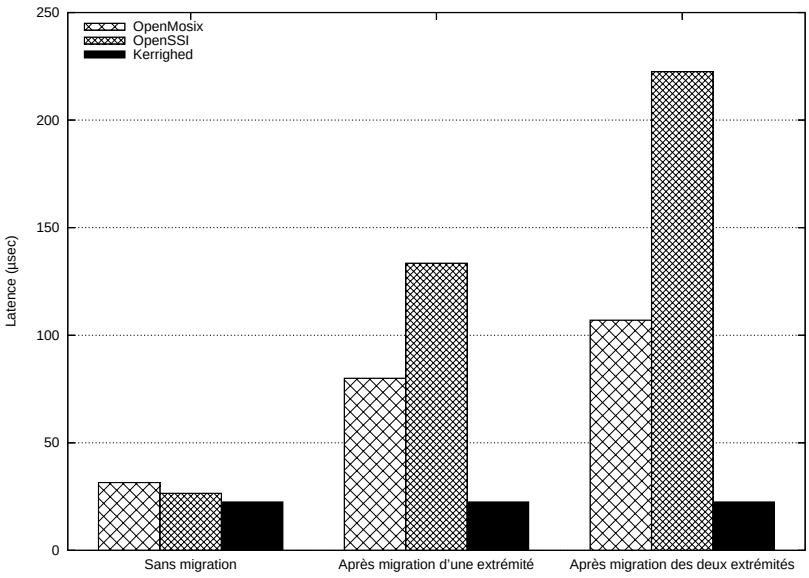

FIG. 5 – Latence des communications par socket TCP, création des extrémités sur les nœuds 1 et 2, migration d'une puis des deux extrémités.

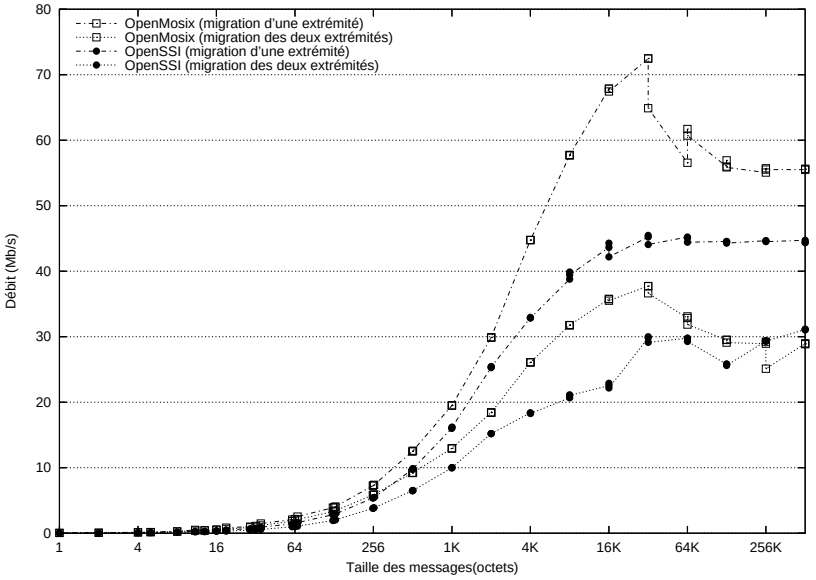

FIG. 6 – Bande passante des communications par socket TCP, création des extrémités sur le nœud 1 (en loopback), puis migration d'une puis des deux extrémités sur le même nœud.

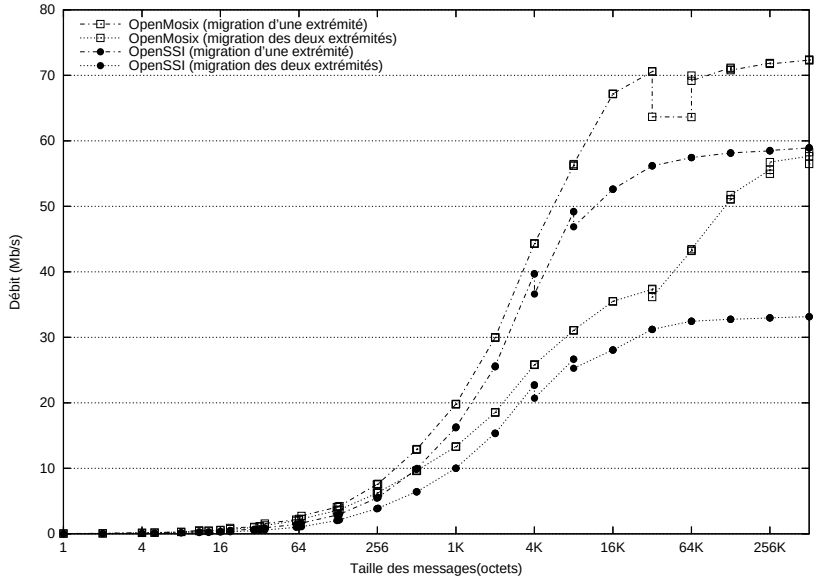

FIG. 7 – Bande passante des communications par socket UNIX, création des extrémités sur le nœud 1, puis migration d'une puis des deux extrémités sur le même nœud.

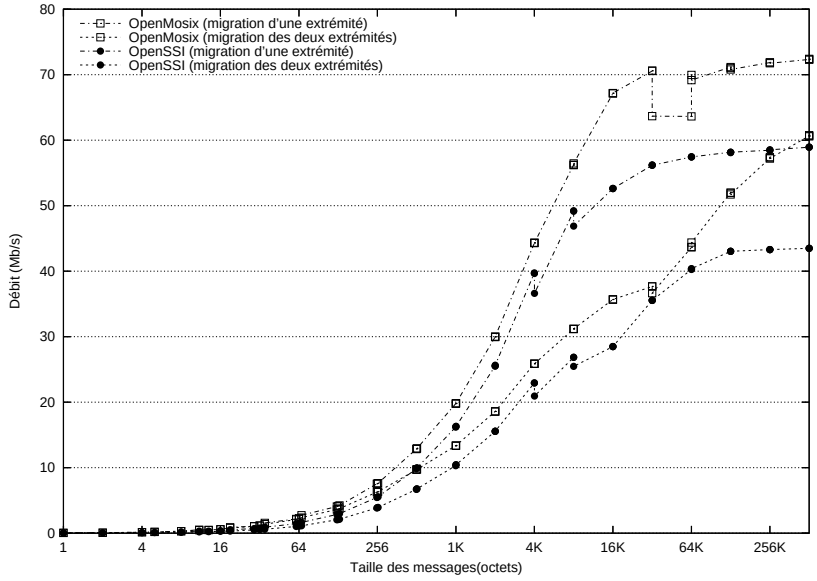

FIG. 8 – Bande passante des communications par socket UNIX, création des extrémités sur le nœud 1, puis migration d'une puis des deux extrémités sur des nœuds différents.

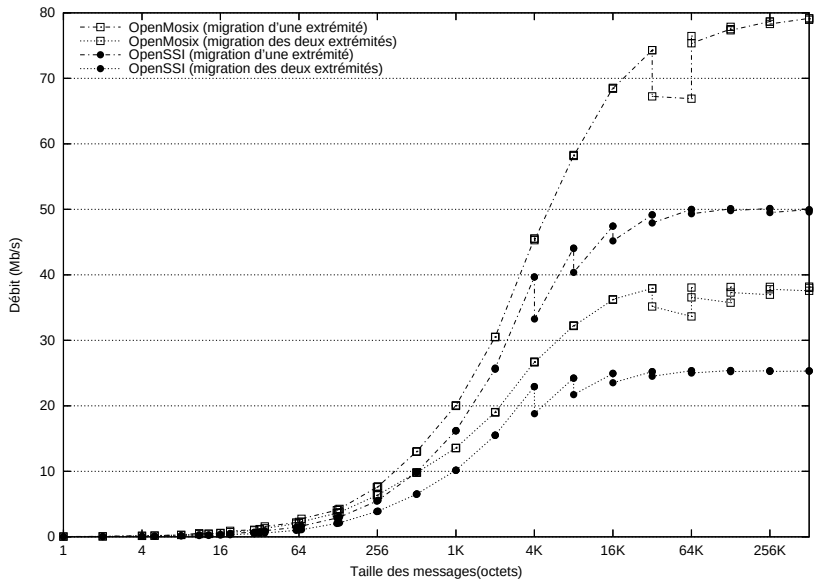

FIG. 9 – Bande passante des communications par tubes, création des extrémités sur le nœud 1, puis migration d'une puis des deux extrémités sur le même nœud.

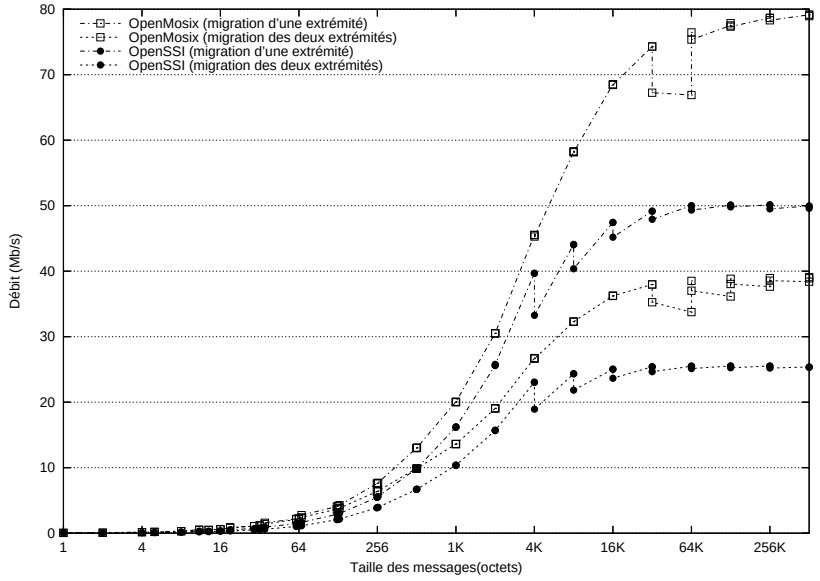

FIG. 10 – Bande passante des communications par tubes, création des extrémités sur le nœud 1, puis migration d'une puis des deux extrémités sur des nœuds différents.

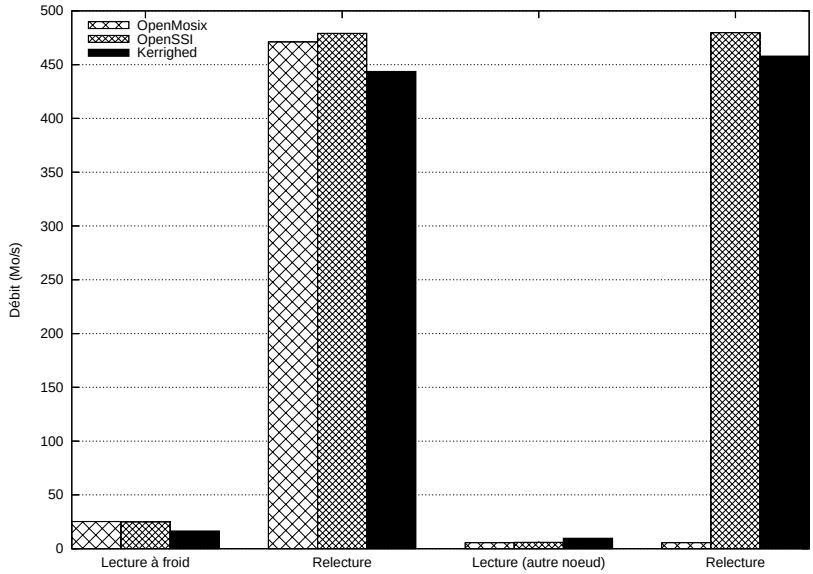

FIG. 11 – Performance du système de fichiers. Lecture à froid puis relecture sur deux nœuds différents.

– Écriture avec migration (figure 13)
  1. écriture à froid d'un fichier sur un nœud
  2. relecture
  3. migration sur un autre nœud puis re-écriture (test de la synchronisation)
  4. relecture

Les résultat du premier et du deuxième test (figures 11 et 12) montrent que OpenMosix, contrairement à OpenSSI et Kerrighed, ne possède pas de cache global. En effet sur un nœud distant, le débit de lecture d'un fichier est toujours celui du réseau.

De plus Kerrighed est plus efficace que les deux autres pour la lecture sur un nœud distant (9 Mo/s contre 5 Mo/s), notamment grâce à l'utilisation d'un système de pré-chargement.

En ce qui concerne l'écriture, Kerrighed a des résultats très faibles pour l'écriture sur un nœud distant après une première écriture (figure 13), la vitesse d'écriture étant de 1.5 Mo/s (contre environ 5.5 Mo/s pour les deux autres). En revanche, pour la relecture, Kerrighed possède à nouveau le cache le plus performant.

Les trois systèmes ont des écritures cohérentes quelque soit le nœd où s'exécute le processus. En revanche, seuls OpenSSI et Kerrighed possèdent un système de cache global.

5 Conclusion

Les trois systèmes proposent les fonctionnalités de base d'un système à image unique : migration des processus et ordonnancement global.

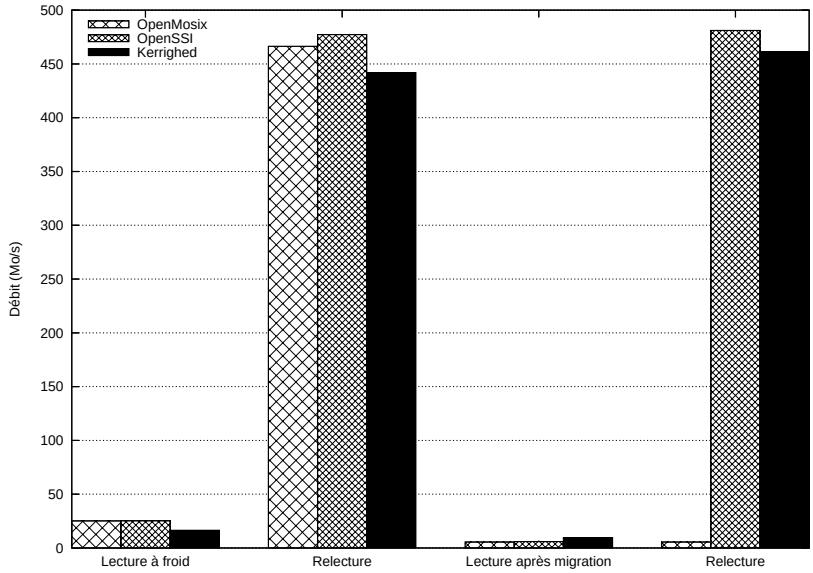

FIG. 12 – Performance du système de fichiers. Lecture à froid puis relecture avant et après migration.

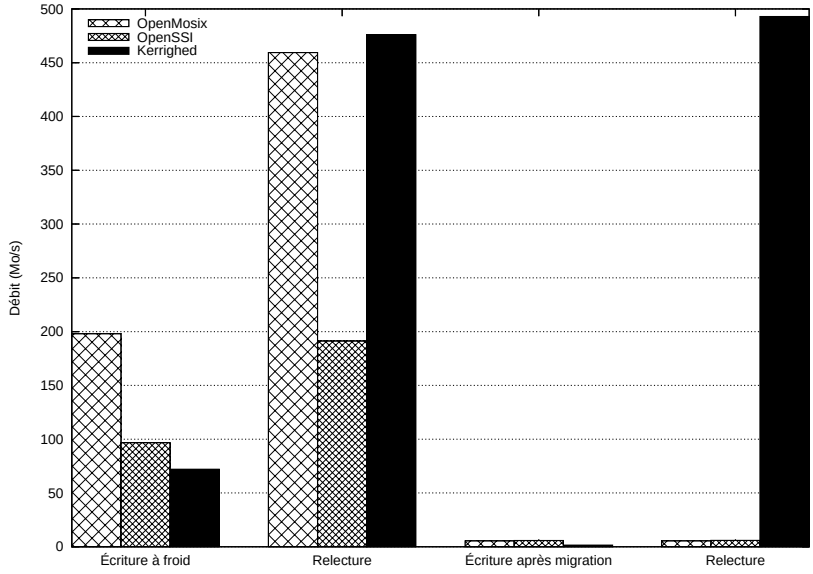

FIG. 13 – Performance du système de fichiers. Écriture puis relecture avant et après migration.

OpenMosix possède le système le plus stable, mais il a aussi le plus de limitations au niveau des fonctionnalités d'image unique (pas de mémoire partagée ou de sémaphores).

OpenSSI est le système proposant le plus de fonctionnalités, mais c'est aussi le système ayant les moins bonnes performances pour les communications inter-processus.

Enfin Kerrighed propose des performances élevées, c'est le seul possédant une mémoire virtuelle partagée distribuée. En revanche c'est le moins stable des trois systèmes, ce qui peut s'expliquer par sa relative jeunesse par rapport aux deux autres systèmes.

Remerciements

Je remercie pour leur aide, leur accueil et leur disponibilité les membres du projet PARIS à l'IRISA et particulièrement Pascal Gallard, Renaud Lottiaux, David Margery, Christine Morin, Louis Rilling, Gaël Utard et Geoffroy Vallée.

Références

[1] M. Bar et MAASK (Maya Anu Asmita Snehal Krushna). Introduction to OpenMosix. 2003. [http ://openmosix.sourceforge.net/linux-kongress-2003_openMosix.pdf]

[2] A. Barak, S. Guday, et R.G. Wheeler. The MOSIX Distributed Operating System, Load Balancing for UNIX. In *Lectures Notes in Computer Science, vol. 672.* Springer-Verlag, 1993.

[3] P. Gallard et C. Morin. Dynamic streams for efficient communications between migrating processes in a cluster. *Parallel Processing Letters*, vol. 13, n°4, pages 930-937. December 2003.

[4] A. M. Goscinski, M. J. Hobbs et J. Silock. Genesis : The operating system managing parallelism and provinding single system image on clusters. Technical report, TR C00/03, School of Computing and Mathematics, Deakin University, February 2000.

[5] R. Lottiaux. *Gestion globale de la mémoire physique d'une grappe pour un système à image unique : mise en œuvre dans le système GOBELINS.* Thèse de doctorat, Université de Rennes 1, IFSIC, France, décembre 2001.

[6] J. K. Ousterhout, A. R. Cherenson, F. Douglis, M. N. Nelson et B. B. Welch. The Sprite network operating system. *Computer*, 21(2) :23-26, February 1988.

[7] E. Pinheiro et R. Bianchini. Nomad : A scalable operating system for clusters of uni and multiprocessors. In *Proceedings of the 1st IEEE International Workshop on Cluster Computing*, December 1999.

[8] M. Schoettner, S. Traub, et P. Schulthess. A transactional DSM Operating System in Java In *Proceedings of the 4th International Conference on Parallel and Distributed Processing Techniques and Applications*, Las Vegas, USA, 1998.

[9] A. S. Tanenbaum, R. van Renesse, and H. van Staveren, G. J. Sharp, S. J. Mullender, J. Jansen et G. van Rossum". Experiences with the Amoeba distributed operating system *Communications of the ACM*, 33(12) :46-63, 1990.

[10] B. Walker et Douglas Steel. Implementing a full single system image unixware cluster : Middleware vs underware. In *Proceedings of International Conference on Parallel and Distributed Processing Techniques and Applications.* PDPTA'99, 1999 :2767-2773.

[11] G. Vallée. *Conception d'un ordonnanceur de processus adaptable pour la gestion globale des ressources d'une grappe de calculateurs : mise en œuvre dans le système d'exploitation Kerrighed.* Thèse de doctorat, Université de Rennes 1, IFSIC, France, 2004.

[12] Manuel de reférence du système Kerrighed [http ://kerrighed.org/docs/PI-1577.pdf]

[13] [http ://www.dragonflybsd.org]

[14] [http ://ssic-linux.sourceforge.net]

[15] [http ://oss.software.ibm.com/dlm]

[16] [http ://linux-ha.org]

[17] [http ://www.linuxvirtualserver.org]