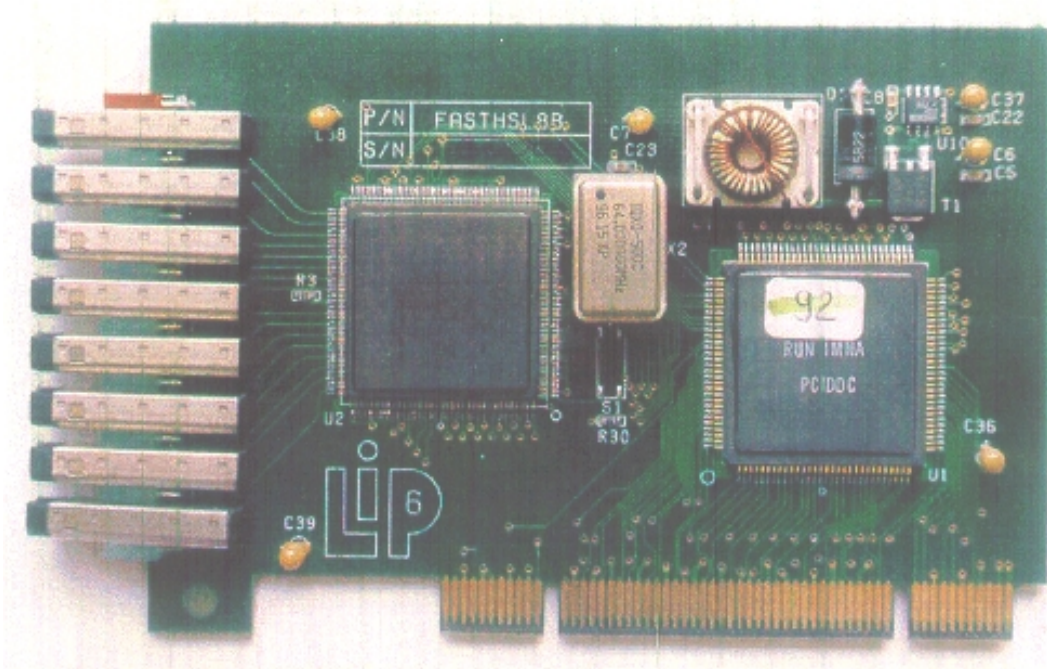


RAPPORT DE STAGE INGENIEUR

NON CONFIDENTIEL



Un système de gestion de tâches pour la machine parallèle MPC

Olivier GLÜCK

Février-Juillet 1999

Sous l'encadrement de Pr. Alain Greiner (Alain.Greiner@lip6.fr)



L'école d'ingénieurs de l'INT

STAGE INGENIEUR 1998 / 1999

Nom du stagiaire : GLÜCK Olivier

Option : Informatique Parallèle et Répartie

Entreprise : Laboratoire Informatique de Paris 6

Dates : Février – Juillet 1999

Sujet : Conception d'un système de gestion de tâches pour une machine parallèle

RESUME :

Ce stage s'inscrit dans le cadre du projet de recherche Multi-PC, qui a démarré en janvier 1995, sous la responsabilité d'Alain GREINER. Il vise la conception et la réalisation d'une machine parallèle à bas coût dont les nœuds de calcul sont des PCs interconnectés par un réseau haut débit.

Le stage a consisté en la définition et la réalisation d'un outil d'administration et d'un système de gestion de tâches pour la machine MPC. Deux interfaces permettent d'utiliser les fonctionnalités de ce logiciel : la ligne de commande UNIX et une interface WWW.

Ce logiciel est constitué d'exécutables écrits en langage C, de shells UNIX, et de programmes CGI.

ABSTRACT :

This training period is a part of the global research project called Multi-PC which started in 1995, under the responsibility of Pr. Alain GREINER. This project deals with the design and the realisation of a low-cost parallel computer composed of calcul-nodes, using standard pentium based PC-motherboards as processing nodes, Unix FreeBSD operating system, and a high speed communication network based on the IEEE 1355/HSL technology.

This training period deals with the definition and the realisation of a Job Management System (JMS), for the parallel computer MPC. Two interfaces were realised : a command-line interface and a WWW interface.

This software is made of C language executables, UNIX shells and CGI programs.

Remerciements

Je tiens à remercier les enseignants - chercheurs, les thésards, et plus généralement toute l'équipe du département ASIM du LIP6 pour le soutien qu'ils ont pu m'apporter, leur ouverture d'esprit, et leur bonne humeur.

Je remercie, en particulier, Alain GREINER pour m'avoir permis d'effectuer ce stage.

Je remercie également, de tout cœur, Daniel MILLOT et Philippe LALEVEE (INT) sans qui ce stage n'aurait pas pu se faire. Je les remercie tout autant, pour toutes les démarches qu'ils ont pu entreprendre à mon égard.

Enfin, je remercie, plus encore, Alexandre FENYO qui m'a encadré et soutenu tout au long de ce stage. Il a toujours été présent pour m'aider à résoudre mes problèmes tout en me laissant par ailleurs, une très grande autonomie de travail.

Table des matières

RAPPORT DE STAGE	1
RÉSUMÉ	2
REMERCIEMENTS	3
TABLE DES MATIÈRES	4
TABLE DES FIGURES	7
CHAPITRE I : INTRODUCTION ET CONTEXTE DU STAGE	9
I. LE LABORATOIRE	9
I.1 L'UNIVERSITÉ DE PARIS 6	9
I.2 LE LABORATOIRE D'INFORMATIQUE DE PARIS 6	10
I.3 LE DÉPARTEMENT ASIM DU LIP6	10
II. LE SUJET DU STAGE	12
II.1 OBJECTIF	12
II.2 DESCRIPTION	12
II.3 MOYENS UTILISÉS	12
II.4 ENCADRANTS	12
III. DESCRIPTION DE LA MACHINE MPC	13
III.1 ARCHITECTURE MATÉRIELLE	13
III.2 ARCHITECTURE LOGICIELLE	14
III.2.1 Le noyau MPC-OS	14
III.2.2 Les drivers MPC	16
III.2.2.1 Le driver CMEM	16
III.2.2.2 Le driver HSL	16
III.2.3 Les démons MPC	17
III.2.4 Les communications via le réseau HSL	17
III.3 L'ENVIRONNEMENT PVM POUR MPC	18
III.3.1 Architecture générale de PVM-MPC	18
III.3.2 Création des tâches de calcul	19
III.3.3 Initialisation de la machine virtuelle PVM-MPC	20
III.3.4 Compilation d'une application PVM-MPC	20

CHAPITRE 2 : LE JMS POUR LA MACHINE MPC **22**

I. GÉNÉRALITÉS SUR LES JMS	22
I.1 LES SYSTÈMES DE GESTIONS DE TÂCHES (JMS)	22
I.2 LES SYSTÈMES EXISTANTS	24
I.3 L'EXEMPLE DE NQE (NETWORK QUEUING ENVIRONNEMENT)	26
II. SPÉCIFICATIONS DU JMS-MPC	27
II.1 POURQUOI UN JMS SPÉCIFIQUE ?	27
II.2 LES RÈGLES DE FONCTIONNEMENT DU JMS POUR MPC	28
II.2.1 Principe général	28
II.2.2 Les queues	29
II.2.3 Le calendrier	31
II.2.4 L'exécutif	32
II.3 LES FONCTIONNALITÉS	33
II.3.1 L'utilisateur	33
II.3.2 L'administrateur	34
II.4 POURQUOI CES CHOIX ?	34
III. RÉALISATION DU JMS-MPC	36
III.1 ARCHITECTURE LOGICIELLE	36
III.1.1 Le répertoire MPC-JMS	36
III.1.2 Les différents types de fichiers	38
III.2 ARCHITECTURE DES FILES D'ATTENTE	39
III.3 L'EXÉCUTIF	40
III.4 LES UTILISATEURS	42
III.5 LES FICHIERS DE CONFIGURATION	42
III.5.1 Le fichier de configuration générale (<i>config</i>)	42
III.5.2 Le fichier de configuration des queues (<i>config_queues</i>)	44
III.5.3 Le fichier de configuration du calendrier (<i>calendar</i>)	44
III.6 QUELQUES POINTS TECHNIQUES	45
III.6.1 Les bibliothèques	45
III.6.2 L'algorithme de tri	48
III.6.3 Les locks	48
III.6.4 La sécurité	49
III.6.4.1 Le bit SUID	49
III.6.4.2 La sécurité et l'interface CGI	49

CHAPITRE 3 : CONCLUSIONS **52**

I. DÉROULEMENT DU STAGE	52
II. APPORTS DU STAGE	52
III. LES RÉUNIONS	53
IV. LES ÉVOLUTIONS DU JMS-MPC	53
V. LE PROJET DE RECHERCHE MPC	53

BIBLIOGRAPHIE **55**

I. SITES INTERNET	55
II. DOCUMENTATIONS ET OUVRAGES	55

ANNEXE 1 : L'ENVIRONNEMENT DE PROGRAMMATION PARALLÈLE PVM 57

I.	INTRODUCTION	57
II.	COMMUNICATIONS	57
III.	FONCTIONNEMENT GÉNÉRAL	57
IV.	VISION CONCEPTUELLE DU SYSTÈME PVM	58

ANNEXE 2 : PVM POUR MPC (COMPLÉMENTS) 59

I.	CRÉATION D'UNE TÂCHE PVM SUR LA MACHINE MPC	59
II.	COMMUNICATION ENTRE UNE TÂCHE DE CALCUL ET UNE TÂCHE MAÎTRE	60
III.	COMMUNICATION ENTRE DEUX TÂCHES DE CALCUL DISTANTES	60
IV.	UN EXEMPLE D'APPLICATION PVM POUR MPC	62
IV.1	CODE SOURCE D'UNE APPLICATION PVM-MPC	62
IV.1.1	Code de master1.c	62
IV.1.2	Code de slave1.c	63
IV.2	COMPILATION D'UNE APPLICATION PVM-MPC	65

ANNEXE 3 : MANUEL D'INSTALLATION DU JMS 66**ANNEXE 4 : GUIDE D'UTILISATION DU JMS 67**

I.	PRÉALABLES	67
II.	L'INTERFACE LIGNE DE COMMANDE	68
II.1	LES COMMANDES UTILISATEUR	68
II.2	LES COMMANDES ADMINISTRATEUR	69
III.	L'INTERFACE CGI	69
III.1	LES COMMANDES UTILISATEUR	69
III.2	LES COMMANDES ADMINISTRATEUR	78

Table des figures

FIGURE 1 : DE L'UPMCP6 AU PROJET MPC	11
FIGURE 2 : ARCHITECTURE DE LA MACHINE MPC DU LIP6.....	13
FIGURE 3 : QUATRE CARTES FASTHSL.....	14
FIGURE 4 : ARCHITECTURE LOGICIELLE DE MPC-OS.....	15
FIGURE 5 : CONTENU INITIAL DE /DEV/HSL.....	16
FIGURE 6 : CONNEXIONS RPC METTANT EN JEU LE NŒUD 0.....	17
FIGURE 7 : PHILOSOPHIE DE PVM-MPC	18
FIGURE 8 : ARCHITECTURE GÉNÉRALE DE PVM-MPC.....	19
FIGURE 9 : VU D'ENSEMBLE DES JMS.....	25
FIGURE 10 : LES COMPOSANTS DU JMS POUR MPC.....	28
FIGURE 11 : LES FILES D'ATTENTE DU JMS.....	29
FIGURE 12 : LE CALENDRIER	31
FIGURE 13 : L'EXÉCUTIF	32
FIGURE 14 : ARBORESCENCE DU JMS-MPC.....	37
FIGURE 15 : LES SOURCES DU JMS.....	38
FIGURE 16 : ÉLÉMENT D'UNE FILE D'ATTENTE	39
FIGURE 17 : ALGORITHME DE L'EXÉCUTIF.....	41
FIGURE 18 : LE FICHIER CONFIG	43
FIGURE 19 : LE FICHIER DE CONFIGURATION DES QUEUEES	44
FIGURE 20 : LE FICHIER CALENDRIER	44
FIGURE 21 : COUCHES DU SYSTÈME PVM.....	58
FIGURE 22 : CRÉATION ET INITIALISATION D'UNE TÂCHE PVM SUR LA MACHINE MPC.....	59
FIGURE 23 : UN EXEMPLE DE COMMUNICATION MAÎTRE-ESCLAVE.....	60
FIGURE 24 : PROTOCOLE DE CONNEXION HSL ENTRE 2 TÂCHES ESCLAVES	61
FIGURE 25 : LE RÉPERTOIRE MPC-JMS SUR LES NŒUDS DE CALCUL.....	67
FIGURE 26 : COMMANDES UTILISATEUR	68
FIGURE 27 : COMMANDES ADMINISTRATEUR	69
FIGURE 28 : PAGE D'ACCUEIL DU JMS	70
FIGURE 29 : MENU UTILISATEUR DU JMS	70
FIGURE 30 : INSÉRER UNE APPLICATION DANS UNE FILE.....	71
FIGURE 31 : RÉSULTAT DE L'INSERTION	72
FIGURE 32 : SUPPRIMER UNE APPLICATION D'UNE QUEUE	72
FIGURE 33 : TUER SON APPLICATION.....	73
FIGURE 34 : ETATS DES FILES D'ATTENTE	74
FIGURE 35 : ETAT DU CALENDRIER.....	74
FIGURE 36 : DEVENIR L'UTILISATEUR PRIVILÉGIÉ	75
FIGURE 37 : VOIR SI UNE APPLICATION EST EN COURS D'EXÉCUTION	75
FIGURE 38 : TESTER L'APPLICATION EN COURS D'EXÉCUTION.....	76
FIGURE 39 : LISTE DES DÉMONS ET DRIVERS	77
FIGURE 40 : MENU ADMINISTRATEUR	78
FIGURE 41 : VIDER UNE FILE D'ATTENTE.....	79
FIGURE 42 : REDÉMARRER LES NŒUDS DE CALCUL	79
FIGURE 43 : CONFIGURER LES FILES D'ATTENTE	80
FIGURE 44 : CONFIGURER LE CALENDRIER	81
FIGURE 45 : DEVENIR UN UTILISATEUR	81



Chapitre I :

Introduction et contexte du stage

I. LE LABORATOIRE	9
I.1 L'UNIVERSITÉ DE PARIS 6	9
I.2 LE LABORATOIRE D'INFORMATIQUE DE PARIS 6	10
I.3 LE DÉPARTEMENT ASIM DU LIP6	10
II. LE SUJET DU STAGE	12
II.1 OBJECTIF	12
II.2 DESCRIPTION	12
II.3 MOYENS UTILISÉS	12
II.4 ENCADRANTS	12
III. DESCRIPTION DE LA MACHINE MPC	13
III.1 ARCHITECTURE MATÉRIELLE	13
III.2 ARCHITECTURE LOGICIELLE	14
III.2.1 Le noyau MPC-OS	14
III.2.2 Les drivers MPC	16
III.2.2.1 Le driver CMEM	16
III.2.2.2 Le driver HSL	16
III.2.3 Les démons MPC	17
III.2.4 Les communications via le réseau HSL	17
III.3 L'ENVIRONNEMENT PVM POUR MPC	18
III.3.1 Architecture générale de PVM-MPC	18
III.3.2 Création des tâches de calcul	19
III.3.3 Initialisation de la machine virtuelle PVM-MPC	20
III.3.4 Compilation d'une application PVM-MPC	20

Chapitre I : Introduction et contexte du stage

Ce chapitre a pour but de se replacer dans le contexte du stage. Il s'agit d'une part de préciser quel a été mon environnement de travail pendant 5 mois et en particulier de situer le laboratoire de recherche qui m'a accueilli par rapport à l'Université Pierre et Marie Curie – Paris 6 (UPMCP6). D'autre part, il s'agit d'explicitier le sujet de mon stage. Enfin, il s'agit de présenter la machine parallèle MPC qui est au cœur de mon stage.

I. Le laboratoire

Mon stage s'est effectué dans le laboratoire de recherche du département *ASIM* du *LIP6* (Laboratoire d'Informatique de Paris 6).

I.1 L'université de Paris 6

Principale héritière de l'ancienne Faculté des Sciences de Paris en Sorbonne, elle forme 3 000 cadres haute technologie par an. L'Université Pierre et Marie Curie est la première Université scientifique et médicale de France.

L'université en quelques chiffres :

- 36 000 étudiants dont 10 000 en 3^{ème} cycle
- 2 600 enseignants – chercheurs
- 1 300 chercheurs
- 150 laboratoires de recherche
- 2 000 DEA ou DESS sont délivrés chaque année (10 % de la totalité française)
- 900 Thèses de doctorat (20 % de la totalité française)

L'université de Paris 6 dispense plusieurs types de formations :

- Formations scientifiques
- Formations médicales
- Formations professionnalisées
- Ecoles d'ingénieurs
- Formation permanente (5 000 stagiaires par an)

Pour plus d'informations : <http://www.admp6.jussieu.fr/>

I.2 Le Laboratoire d'Informatique de Paris 6

Le LIP6 est l'instrument de la politique scientifique de l'université Pierre et Marie Curie en matière d'informatique, exprimée dans le cadre du contrat quadriennal 1997-2000 qui lie l'université, le CNRS et le ministère.

Le LIP6 en quelques chiffres :

- 9 thèmes de recherche
- Un effectif d'environ 320 personnes (doctorants compris)
- 114 enseignants-chercheurs et chercheurs permanents
- 165 thésards et post-docs

Le laboratoire se compose de 9 thèmes de recherche, dont les activités recouvrent une large part de l'informatique :

- Algorithmique numérique et parallélisme (ANP)
- Apprentissage et acquisition de connaissances (APA)
- *Architecture des systèmes intégrés et micro-électronique (ASIM)*
- Calcul formel (CALFOR)
- Objets et Agents pour Systèmes d'Information et de Simulation (OASIS)
- Réseaux et performances (RP)
- Sémantique, preuve et implantation (SPI)
- Systèmes répartis et coopératifs (SRC)
- Systèmes d'aide à la décision et à la formation (SYSDEF)

Outre leurs activités propres, ces thèmes collaborent dans diverses actions transversales et sont engagés dans de nombreux projets en partenariat avec des entreprises.

Huit projets transversaux déterminent la politique scientifique du LIP6. La machine MPC fait partie de ces projets.

Pour plus d'informations : <http://www.lip6.fr/>

Mon stage s'est effectué au sein du département *ASIM* du LIP6.

I.3 Le département ASIM du LIP6

Le département ASIM concentre ses recherches dans les domaines suivants :

- La conception de circuits intégrés hautement complexes
- Le développement d'outils CAO avancés pour la VLSI
- L'architecture matérielle des systèmes.

Son effectif est de 65 personnes environ.

Les enseignements dispensés par le département sont :

- D.E.A en Architecture des Systèmes Intégrés et Micro-Electronique (ASIME)
- DESS de Circuits Intégrés et Systèmes Analogiques Numériques
- Maîtrise Informatique (Option Architecture) et Maîtrise EEA (Option MEMI)

La recherche du département se décompose principalement en trois projets :

- La machine MPC
- Projet d'indexation multimédia
- Projet CAO de circuits et systèmes (Alliance)

Alliance est un outil avancé de CAO pour la VLSI. Les activités de l'indexation Multimédia sont l'extraction d'informations de contenu, la recherche d'informations et les ontologies et la représentation de connaissances.

Le projet MPC est un projet de longue haleine. L'idée générale est d'aider certains demandeurs de puissance de calcul du LIP6 à mener des expériences sur la machine MPC développée au laboratoire dans le thème ASIM. Mais surtout, le but du projet MPC est la conception d'une machine parallèle performante et à faible coût. La contribution du laboratoire s'est traduite par l'acquisition d'un réseau à 4 nœuds. Ceci est suffisant pour mettre au point des expériences mais il serait bon d'avoir au moins huit nœuds pour obtenir des résultats significatifs.

Mon stage est au cœur de ce projet comme le précise la figure ci-dessous (*Figure 1*). Je m'attacherai donc, dans la section III à la description de la machine MPC du LIP6 (cf. *Chapitre 1 : Description de la machine MPC*).

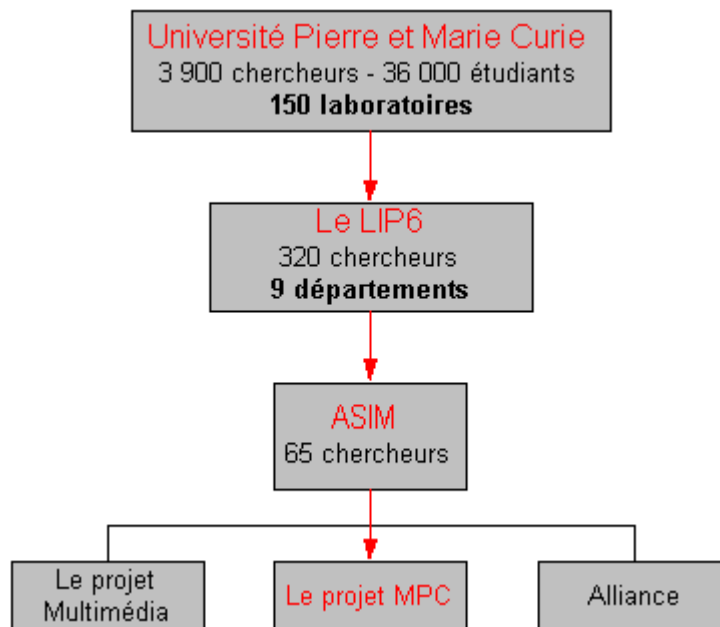


Figure 1 : De l'UPMCP6 au projet MPC

Pour plus d'informations : <http://www-asim.lip6.fr/>

II. Le sujet du stage

II.1 Objectif

L'objectif du stage est de mettre en place un système d'administration automatisé de la machine parallèle MPC du laboratoire LIP6, ainsi que de l'environnement PVM de développement fourni avec celle-ci.

II.2 Description

L'outil d'administration de la machine MPC doit, d'une part pouvoir gérer le chargement des couches logicielles et d'autre part permettre aux différents utilisateurs de tester leurs applications.

L'environnement standard de programmation distribuée PVM a été porté sur l'interface de programmation proposée par les couches de communications MPC.

Les couches logicielles qui forment le noyau de communication spécifique à la machine MPC sont constituées des drivers et démons qui assurent l'interface avec la carte réseau *FastHSL* et des couches de communication PVM associées à ces drivers (cf. [Chapitre 1 : III.2 Architecture logicielle](#)). Avant l'activation d'une nouvelle tâche, une réinitialisation des différentes couches est prévue.

Différents utilisateurs pourront utiliser la machine MPC simultanément. Il s'agit d'assurer le bon fonctionnement de cet environnement multi-utilisateurs tout en considérant qu'une seule application PVM à la fois pourra être lancée sur les nœuds de calcul.

II.3 Moyens utilisés

Les moyens utilisés sont la machine MPC du LIP6 avec l'ensemble des couches logicielles associées (MPC et PVM). La machine est composée de 4 nœuds de calcul et d'une console (cf. [Chapitre 1 : III.1 Architecture matérielle](#)).

II.4 Encadrants

Directeur de stage : **Alexandre Fenyo**, Laboratoire d'informatique de Paris 6
Conseiller d'études : **Philippe Lalevée**, enseignant-chercheur à l'INT

III. Description de la machine MPC

Le projet MPC est un des projets les plus importants du département ASIM du LIP6. Il a pour but la conception d'une machine parallèle performante à faible coût.

III.1 Architecture matérielle

La machine MPC du LIP6 est une machine parallèle de type « grappe de PCs ». Elle est constituée d'un ensemble de PCs standard interconnectés par un réseau à très haut débit dont la technologie a été développée par le laboratoire LIP6. Elle est composée de 4 nœuds de calcul Bi-Pentium haut de gamme et d'une console pour améliorer l'exploitation de la machine (cf. *Figure 2*).

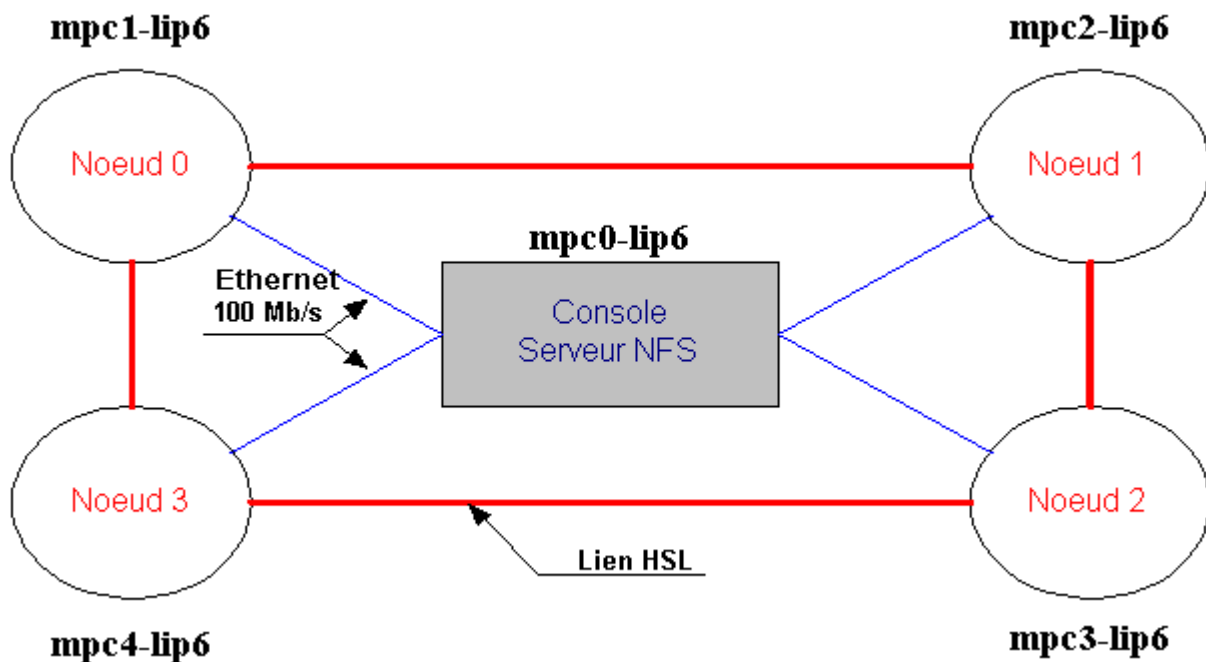


Figure 2 : Architecture de la machine MPC du LIP6

Le réseau d'interconnexions rapides est un réseau à 1 Gigabit/s en full duplex à la norme IEEE 1335 HSL. Il est composé de **liens HSL** (câbles coaxiaux) et d'une **carte FastHSL** sur chaque nœud. Tous les nœuds sont également équipés d'une carte Ethernet pour constituer un réseau de contrôle non seulement, entre eux, mais aussi avec la console.

La carte FastHSL a été conçue au laboratoire LIP6. Elle contient, en particulier, 2 circuits VLSI (cf. *Figure 3*):

- Un contrôleur de bus PCI intelligent appelé **PCI-DDC**
- Un routeur rapide possédant 8 liens HSL à 1 Gbit/s appelé **Rcube**

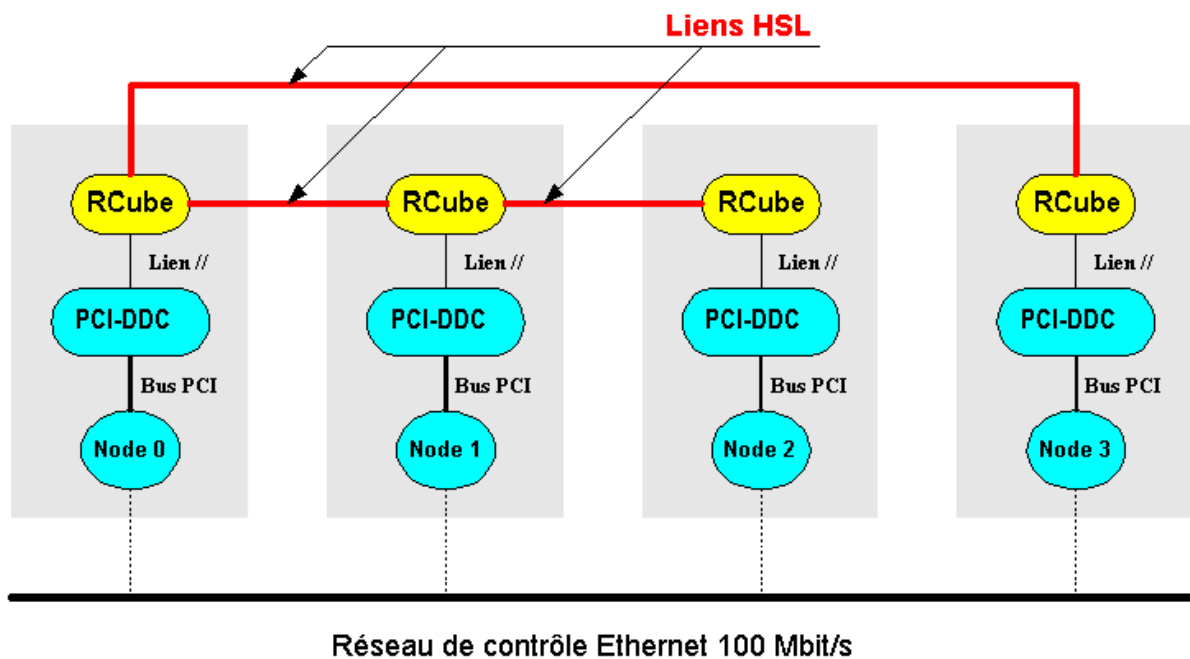


Figure 3 : Quatre cartes FastHSL

PCI-DDC réalise le protocole de communication et **RCUBE** le routage des paquets.

Le réseau ainsi constitué fournit une primitive de communication extrêmement efficace d'écriture en mémoire distante, qui peut être assimilée à un « Remote DMA ». L'enjeu est de faire bénéficier les applications de la très faible latence matérielle du réseau, en minimisant le coût des couches logicielles. MPC promet donc de très bonnes performances via son réseau d'interconnexion haut-débit.

III.2 Architecture logicielle

La machine MPC du LIP6 fonctionne actuellement sur UNIX FreeBSD 3.1. Une pile de drivers noyaux et de démons forment les couches de communication MPC. Cet ensemble porte le nom de noyau MPC-OS. Il sera bientôt disponible sous LINUX.

III.2.1 Le noyau MPC-OS

Le noyau MPC-OS est l'ensemble des couches logicielles qui permettent d'utiliser au mieux les performances matérielles de la carte FastHSL. Elles permettent également un accès à différents niveaux pour les utilisateurs et développeurs (cf. *Figure 4*).

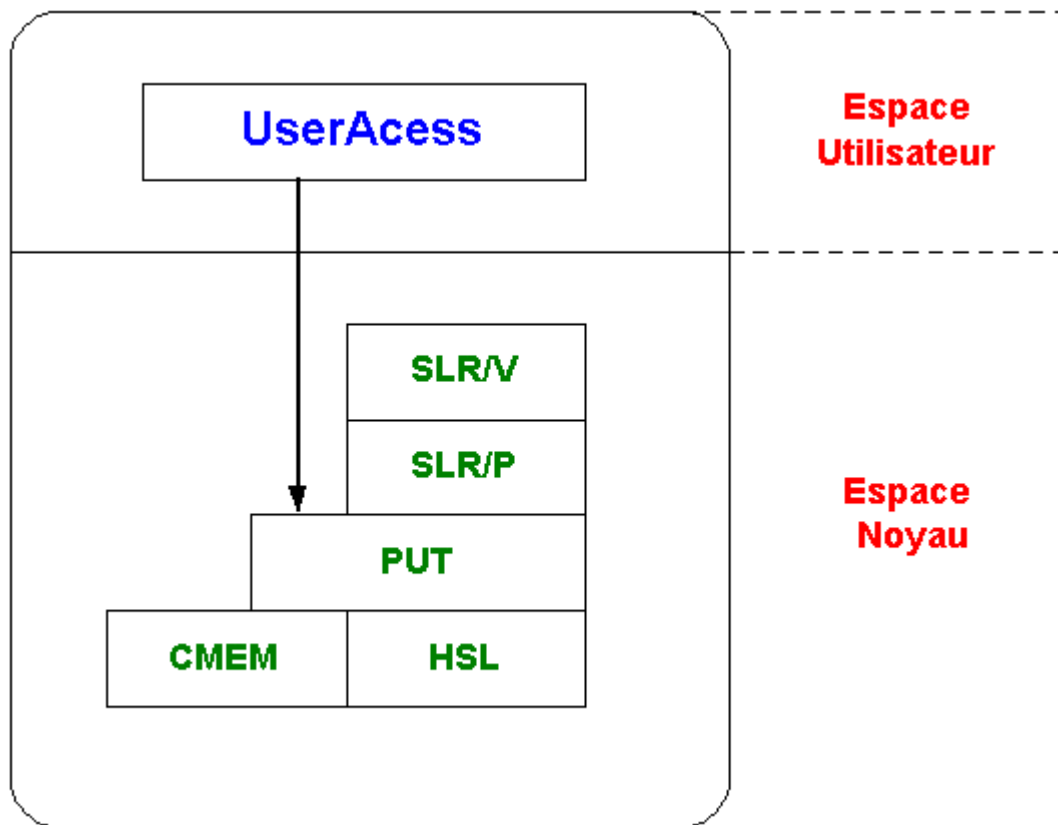


Figure 4 : Architecture logicielle de MPC-OS

La description de ces différentes couches est la suivante :

- La couche PUT constitue le service de communication de plus bas niveau et fournit la fonction *remote write* de l'espace d'adressage physique local vers l'espace physique d'un nœud distant.
- La couche SLRP fournit un service d'échange de zone de mémoire physique, entre nœuds, sur des canaux virtuels.
- La couche SLRV fournit un service équivalent mais en manipulant des zones localisées dans des espaces de mémoire virtuel.
- La bibliothèque UserAcces permet d'accéder aux fonctionnalités de la couche PUT depuis un processus en mode utilisateur.
- CMEM et HSL sont les deux drivers MPC qui permettent d'accéder aux fonctionnalités de la carte FastHSL.

On distingue quatre niveaux d'accès aux couches logicielles MPC :

- Accès depuis le noyau à SLRV
- Accès depuis le noyau à SLRP
- Accès depuis le noyau à PUT
- Accès depuis un processus à PUT (USRPUT)

Ainsi, le portage d'une application peut se faire à différents niveaux suivants les fonctionnalités souhaitées. Par exemple, le portage de l'environnement de programmation

PVM a été réalisé au niveau de la couche SLR/V (cf. *Chapitre 1 : III.3 L'environnement PVM pour MPC*).

Une version de PUT en mode utilisateur est en cours de développement au PRISM (Versailles) qui portera le nom « PAPI ».

III.2.2 Les drivers MPC

L'architecture logicielle des services de communication de la machine MPC est composée, entre autres, de 2 modules (ou drivers) noyaux chargeables dynamiquement.

III.2.2.1 Le driver CMEM

Ce pilote est accessible par le pseudo-périphérique `/dev/cmем`. Cmem est un gestionnaire de mémoire physique. Il se réserve une plage de mémoire contiguë qu'il mappe dans l'espace d'adressage virtuel du noyau ; il est capable par la suite de fournir des morceaux de cette plage aux modules qui en feront la demande (comme par exemple *hsldriver*).

Il est chargé dynamiquement dans le noyau lors du démarrage de la machine pour qu'il puisse trouver suffisamment de mémoire physique contiguë libre afin de couvrir ses besoins.

III.2.2.2 Le driver HSL

Il constitue la couche de plus bas niveau. Il gère le pseudo-périphérique `/dev/hsl` et pilote le matériel.

Il contient le code des couches suivantes :

- SLR/V
- SLR/P
- PUT
- HSLDRIVER (pseudo-périphérique en mode caractère)

Le module HSL Driver fait appel à des procédures de CMEM Driver car il utilise la mémoire allouée par CMEM.

Une fois le module HSL chargé, le contenu de `/dev/hsl` permet de suivre l'état des communications HSL.

```

/dev/hsl : kernel drivers and protocols for MPC parallel computer

***** SLR/V - SCP/V
calls to send()           :      0
calls to recv()          :      0
***** SLR/P - SCP/V
calls to send()           :      0
calls to recv()          :      0
calls to ddslrp_timeout() :      0
timeouts from timeouts   :      0
***** PUT
calls to put_add_entry()  :      0
hardware interrupts      :      0
    
```

Figure 5 : Contenu initial de /dev/hsl

Afin de pouvoir utiliser les primitives de communications HSL, il faut également lancer les deux démons.

III.2.3 Les démons MPC

Deux démons sont nécessaires pour utiliser le réseau MPC. Ce sont les exécutables **hslclient** et **hslserver**. Ils sont construits au-dessus de la couche RPC et dialoguent entre eux via le réseau de contrôle (Ethernet). Ils se chargent de la configuration des couches logicielles du noyau des différents nœuds de la machine MPC. Chaque client consulte un fichier de configuration local au nœud indiquant le nom de domaine de chacun des autres nœuds et contacte les serveurs qui y résident. Il faut donc activer sur chaque nœud les deux démons dans un ordre quelconque. Le démon hslclient utilise également un fichier de configuration qui précise les tables de routages du réseau HSL sur chaque nœud.

La figure ci-dessous montre les connexions qui s'établissent entre le nœud 0 et tous les autres nœuds. Ces connexions sont des RPC (Remote Procedure Call).

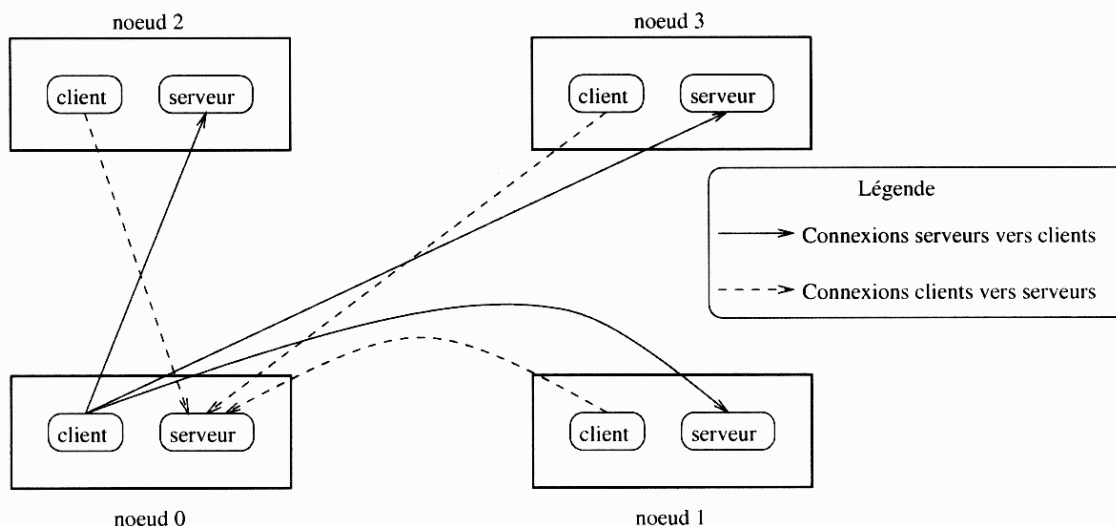


Figure 6 : Connexions RPC mettant en jeu le nœud 0

III.2.4 Les communications via le réseau HSL

Le modèle de communication est le passage de messages sous le mode *Receiver-Driven*. Autrement dit, le transfert du message se déroule sous le contrôle du récepteur. Il est initié par celui-ci en exécutant la primitive *slrpv_receive*. Celle-ci spécifie au système les zones mémoires destinées à la réception du message et provoque l'émission d'un message de contrôle qui invite l'émetteur à transférer ses données dans la zone mémoire spécifiée par le récepteur. L'émetteur effectue de son côté un appel à la primitive *slrpv_send*.

La communication dans MPC suppose l'existence d'une information commune à l'émetteur et au récepteur. Il s'agit du canal de communication qui sert à distinguer les destinataires potentiels d'un message se trouvant sur un même nœud.

Pour plus d'informations : <http://www-asim.lip6.fr/mpc>

III.3 L'environnement PVM pour MPC

Parmi les environnements de programmation parallèle, c'est l'environnement standard PVM (Parallel Virtual Machine) qui a été choisi pour la machine MPC. Quelques explications sur cet environnement sont disponibles à *l'Annexe 1 : L'environnement de programmation parallèle PVM*.

Il a donc été nécessaire de faire une implémentation de PVM au-dessus des couches de communication MPC. Le niveau de portage choisi a été la couche de communication SLR/V qui permet d'échanger via le réseau HSL des zones de mémoire virtuelle sur des canaux virtuels.

L'objectif d'une implémentation de PVM sur une machine multiprocesseur comme la machine MPC est d'explorer les potentialités d'un réseau rapide pour améliorer les performances des systèmes PVM. Par ailleurs, l'utilisation d'une librairie à passage de messages comme PVM facilite le développement d'applications utilisateur.

L'implémentation est basée sur la version 3.3 de PVM et est optimisée pour l'architecture de la machine MPC. Elle n'utilise qu'un seul démon pour toute la machine virtuelle. Celui-ci initialise la machine virtuelle et assure la gestion de toutes les tâches comme si elles étaient locales à son nœud.

III.3.1 Architecture générale de PVM-MPC

Le nœud sur lequel tourne le démon est appelé nœud de service ou bien nœud de démarrage (il joue le rôle d'un frontal). Ce nœud doit être le nœud 0. Toutes les applications PVM-MPC devront être lancées sur le nœud de service. Les autres nœuds MPC sont des nœuds de calcul pour la machine virtuelle (voir figure). Cela permet de voir toute la machine MPC comme une seule machine gérée par un démon unique.

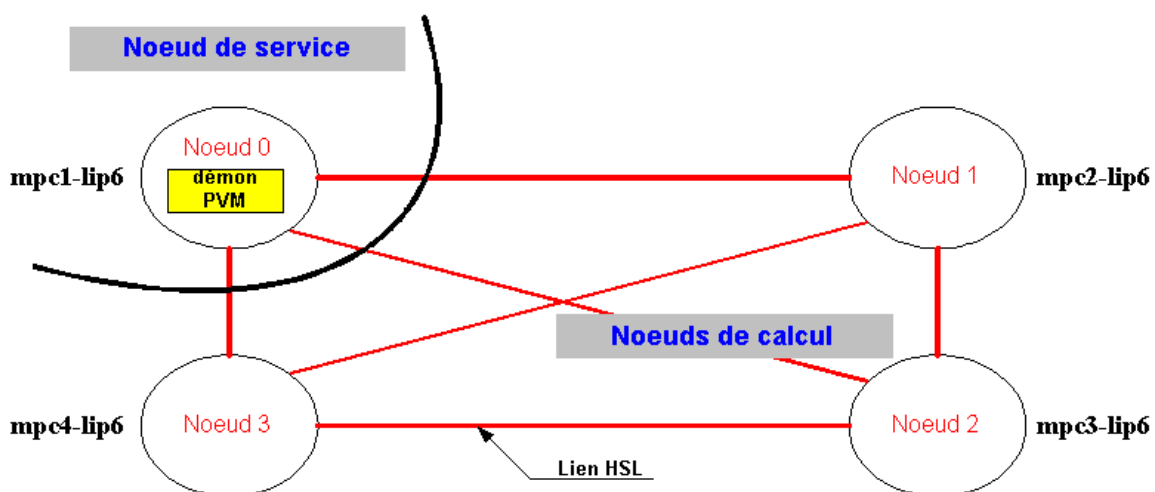


Figure 7 : Philosophie de PVM-MPC

Dans cette implémentation, une première tâche peut être lancée sur le nœud de démarrage et intégrer la machine virtuelle par l'établissement d'une connexion *Unix Socket* avec le démon. Dans la suite, nous appellerons cette tâche la tâche maître. Les tâches lancées par le démon sur le nœud de calcul seront les tâches esclaves. **Par ailleurs, la seule manière de lancer les tâches sur les nœuds de calcul est d'effectuer un *pvm_spawn()* via la tâche maître, sur le nœud de démarrage.** Le choix des nœuds de calcul devant supporter les tâches est laissé à la charge du démon pour un meilleur rendement. Ceci implique que, en particulier, la fonction *pvm_spawn()* a été modifié par rapport au PVM standard. Son appel est légèrement différent.

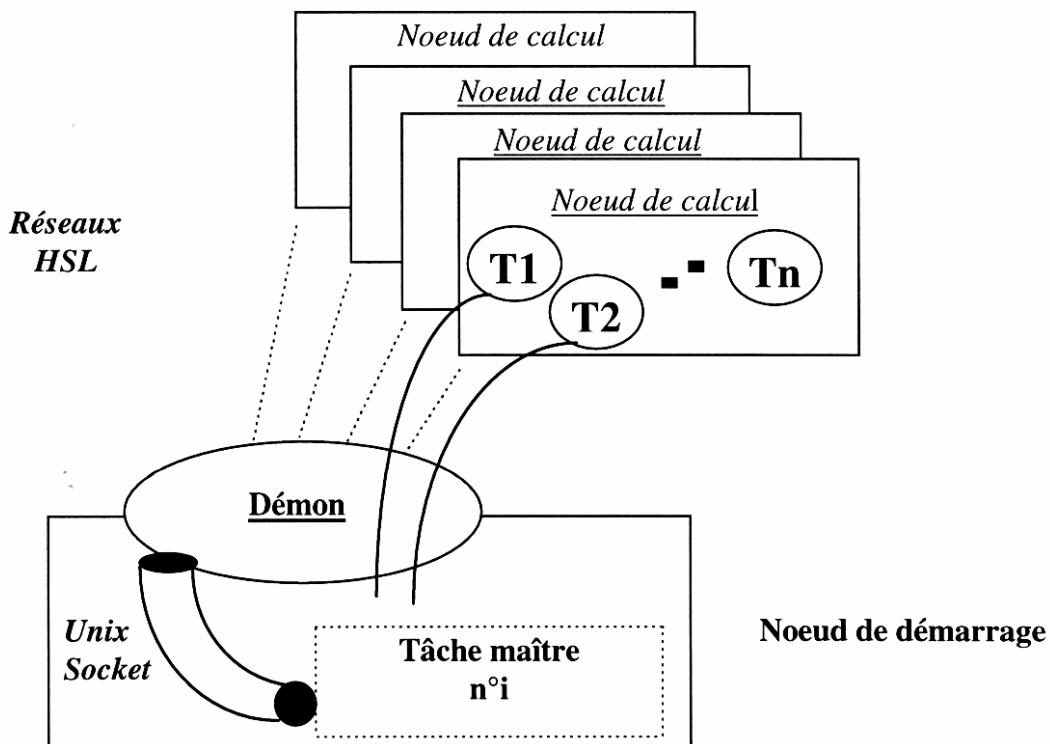


Figure 8 : Architecture générale de PVM-MPC

III.3.2 Création des tâches de calcul

Les tâches de calcul sont créées grâce à l'appel dans la tâche maître de la procédure *pvm_spawn()* (cf. [Figure 22](#)). Le démon se charge de la répartition de ces tâches sur les nœuds de calcul. La tâche de calcul est créée via l'exécution, par le démon, d'un *rsh* (remote shell) sur le nœud distant choisi pour l'accueillir. Le démon se met alors en attente jusqu'à la réception d'une demande de contexte de la part de la tâche lancée. Cette demande est produite lors de l'exécution d'un appel à la bibliothèque PVM. Le contexte de la tâche, contenant entre autres son tid (task identifier géré par PVM), lui est alors communiqué. Le tid est le seul moyen pour le démon de distinguer les tâches tournant sur la machine virtuelle. Dès lors, une connexion hsl est établie entre cette tâche et le démon. Cette connexion est maintenue tout au long de la durée de vie de cette tâche au sein de la machine virtuelle. Elle servira aux

échanges de messages de contrôle avec le démon ainsi qu'aux échanges de données avec la tâche maître. Un canal de communication est négocié pour chaque nœud MPC en fonction du nombre de tâches déjà présentes sur ce dernier.

Pour plus de détails sur PVM pour MPC, consulter [l'Annexe 2 : PVM pour MPC \(compléments\)](#).

III.3.3 Initialisation de la machine virtuelle PVM-MPC

Avant d'initialiser la machine virtuelle, il est nécessaire d'initialiser le réseau HSL de la machine MPC comme cela a été décrit dans la section [Chapitre 1 : III.2 Architecture logicielle](#). Nous supposons donc que les pilotes HSLDRIVER et CMEMDRIVER sont chargés et que les démons *hslclient* et *hslserver* sont lancés sur tous les nœuds.

Un autre pilote doit être chargé sur tous les nœuds pour que PVM-MPC puisse fonctionner. Il s'agit du pilote PVMDRIVER. Ce driver sert d'interface entre les communications PVM et les couches logicielles MPC. En particulier, le driver PVMDRIVER permet d'accéder aux primitives de la couche SLR/V.

La commande Unix `modstat` doit faire apparaître les 3 pilotes nécessaires à l'utilisation de PVM-MPC :

Type	Id	Off	Loadaddr	Size	Info	Rev	Module Name
MISC	0	0	f5568000	0008	f5569000	1	daemon_saver_mod
DEV	1	128	f5571000	0013	f5572038	1	cmemdriver_mod
DEV	2	129	f7576000	05aa	f7586058	1	hsldriver_mod
DEV	3	130	f76e4000	0008	f76e5038	1	pvmdriver_mod

Enfin, il faut lancer le démon PVM sur le nœud de service (nœud 0) avant de lancer des applications parallèles PVM. Désormais, la machine virtuelle PVM-MPC est initialisée.

III.3.4 Compilation d'une application PVM-MPC

L'implémentation de PVM-MPC est, rappelons le, à démon unique. Ceci implique que deux comportements différents sont possibles pour les tâches suivant qu'elles se trouvent sur le nœud du démon ou sur les nœuds de calcul. Deux bibliothèques différentes sont utilisées suivant le cas :

- Les tâches maîtres doivent être liées avec la bibliothèque *libpvm3.a* (bibliothèque standard de PVM)
- Les tâches esclaves doivent être liées avec la bibliothèque *libpvm3pe.a* (bibliothèque PVM-MPC)

Le maître utilise la bibliothèque PVM standard. Un exemple de compilation d'une application PVM pour MPC est donné en [Annexe 2 : IV.2 Compilation d'une application PVM-MPC](#).

Chapitre II :

Le JMS pour la machine MPC

I. GÉNÉRALITÉS SUR LES JMS	22
I.1 LES SYSTÈMES DE GESTIONS DE TÂCHES (JMS)	22
I.2 LES SYSTÈMES EXISTANTS	24
I.3 L'EXEMPLE DE NQE (NETWORK QUEUING ENVIRONNEMENT)	26
II. SPÉCIFICATIONS DU JMS-MPC	27
II.1 POURQUOI UN JMS SPÉCIFIQUE ?	27
II.2 LES RÈGLES DE FONCTIONNEMENT DU JMS POUR MPC	28
II.2.1 Principe général	28
II.2.2 Les queues	29
II.2.3 Le calendrier	31
II.2.4 L'exécutif	32
II.3 LES FONCTIONNALITÉS	33
II.3.1 L'utilisateur	33
II.3.2 L'administrateur	34
II.4 POURQUOI CES CHOIX ?	34
III. RÉALISATION DU JMS-MPC	36
III.1 ARCHITECTURE LOGICIELLE	36
III.1.1 Le répertoire MPC-JMS	36
III.1.2 Les différents types de fichiers	38
III.2 ARCHITECTURE DES FILES D'ATTENTE	39
III.3 L'EXÉCUTIF	40
III.4 LES UTILISATEURS	42
III.5 LES FICHIERS DE CONFIGURATION	42
III.5.1 Le fichier de configuration générale (<i>config</i>)	42
III.5.2 Le fichier de configuration des queues (<i>config_queues</i>)	44
III.5.3 Le fichier de configuration du calendrier (<i>calendar</i>)	44
III.6 QUELQUES POINTS TECHNIQUES	45
III.6.1 Les librairies	45
III.6.2 L'algorithme de tri	48
III.6.3 Les locks	48
III.6.4 La sécurité	49
III.6.4.1 Le bit SUID	49
III.6.4.2 La sécurité et l'interface CGI	49

Chapitre 2 : Le JMS pour la machine MPC

Ce chapitre a pour but de décrire l'outil d'administration automatisé de la machine parallèle MPC du laboratoire LIP6 qui a été conçu durant le stage. Cet outil porte le nom de « **JMS pour la machine MPC (JMS-MPC)** ». Le terme « JMS » vient de l'anglais *Job Management System* qui signifie Système de gestion de tâches. Je m'attacherai tout d'abord à donner quelques généralités sur les JMS existants dans le monde du parallélisme dont je me suis inspiré. Deuxièmement, j'expliquerai les solutions choisies pour la machine MPC. Enfin, je rentrerai plus en détails dans la réalisation finale de cet outil d'administration et de gestion de tâches.

I. Généralités sur les JMS

Le regroupement de plusieurs machines sur un réseau ne suffit pas à la constitution d'un véritable *cluster*. Un logiciel assurant la gestion de l'ensemble est nécessaire à la meilleure utilisation possible de la puissance de calcul des ces nœuds indépendants. Un système de gestion de tâches a justement pour but de gérer cette puissance de calcul afin d'avoir les meilleures performances possibles et de permettre à plusieurs utilisateurs de lancer des tâches simultanément.

I.1 Les systèmes de gestions de tâches (JMS)

Les systèmes de gestion de tâches s'utilisent sur diverses machines parallèles ou *clusters* dans le monde entier. Ils disposent généralement des fonctionnalités suivantes :

- Une interface utilisateur
- Un *scheduler*
- Un gestionnaire des ressources
- Rapatriement des fichiers de log et de configuration sur une même machine
- Un environnement sécurisé

L'**interface utilisateur** permet d'exécuter des tâches locales ou distantes. Celles-ci sont lancées par l'intermédiaire de queues (ou files d'attente). L'utilisateur peut :

- Spécifier les ressources nécessaires aux tâches qu'il veut exécuter
- Supprimer l'exécution d'une tâche
- Faire de la suspension ou de la reprise d'exécution de tâche
- Connaître le statut d'une tâche en cours d'exécution

Pour ce faire, l'utilisateur dispose suivant les cas d'une ligne de commande, d'un environnement graphique ou bien des deux interfaces simultanément.

Le scheduler sert à définir une politique de scheduling, c'est à dire de lancement des tâches. En effet, celles-ci doivent être classées les une par rapport aux autres selon des priorités à définir. Ces priorités sont calculées généralement selon les critères suivants :

- Temps d'attente dans les queues avant exécution
- Ressources nécessaires (nombre de nœuds, temps d'exécution, mémoire, disque, etc.)
- Type de la tâche (interactive, parallèle, batch, etc.)
- Identité de l'utilisateur

Les priorités peuvent être *statiques* ou *dynamiques* suivant que celles-ci soient fixées une fois pour toutes lors de la mise en queue ou qu'elles évoluent dans le temps. Par exemple, *dans le cas de priorités statiques*, une façon courante de procéder est d'appliquer la politique « first-in, first-out ». Le premier job arrivé est le premier job à être exécuté. Une autre façon classique de procéder est d'attribuer différentes priorités aux différents utilisateurs. *Dans le cas de priorités dynamiques*, il est par exemple possible de privilégier, dans la journée, les tâches interactives ou celles dont l'exécution est courte.

Le gestionnaire des ressources sert à allouer des ressources, à connaître l'état des ressources, à collecter toutes sortes d'informations relatives à l'exécution des tâches et à partir de celles-ci à appliquer la politique de scheduling.

Par ailleurs, certains systèmes offrent d'autres fonctionnalités comme par exemple la migration de tâches ou la reprise d'exécution à des points de synchronisation antérieurs.

Il y a principalement trois modes d'utilisation d'un JMS :

- *Dedicated mode* : un seul utilisateur ne peut lancer qu'une seule tâche à la fois
- *Space sharing* : plusieurs tâches peuvent s'exécuter en même temps mais sur des partitions de nœuds disjointes
- *Time sharing* : plusieurs tâches peuvent s'exécuter simultanément sur un même nœud.

Un JMS gère souvent de nombreux types d'applications : les applications de type batch, les applications interactives, les applications avec reprise sur points de synchronisation et les applications parallèles.

Un JMS doit aussi définir des limites d'utilisation des machines pour des classes d'utilisateurs à définir. Par exemple, si besoin est, certains utilisateurs peuvent bénéficier de priorités plus élevées pour lancer leur applications. Cela se fait très souvent à l'aide de files d'attente. Il est alors possible de définir des propriétaires, des gestionnaires et des simples utilisateurs de ces queues.

I.2 Les systèmes existants

Il existe plus d'une vingtaine de JMS de nos jours. Certains sont publics ; d'autres sont payants. Les 4 Job Management System les plus connus sont :

- **DQS et Codine** : <http://www.msi.umn.edu/bscl/info/dqs/>

Distributed Queuing System (DQS) fait partie du domaine public. Il est beaucoup utilisé et à désormais une version commerciale appelée *Codine*. GENIAS commercialise ce produit en Europe.

- **LSF** : http://www.lerc.nasa.gov/WWW/LSF/lsf_homepage.html

Load Sharing System (LSF), développé au Canada, est probablement le JMS le plus utilisé avec plus de 20 000 licences vendues dans le monde entier.

- **NQE** : <http://www.sgi.com/software/nqe/>

Network Queuing Environnement (NQE) est vendu par la compagnie *Cray* qui a conçu ce JMS pour les machines parallèles que nous lui connaissons. Ce système sera étudié à la section *Chapitre 2 : I.3 L'exemple de NQE (Network Queuing Environnement)*

- **Condor et NQS** : <http://www.cs.wisc.edu/condor/>

Ces deux JMS ont également une grande influence. Condor est du domaine public et est l'un des premiers JMS à supporter la migration des processus et la reprise sur point de synchronisation. NQS a été développé dans les années 1980 en collaboration avec la NASA.

Récemment, le JMS *GNU Queue* a fait son apparition.

Le tableau *Figure 9* présente les fonctionnalités des 4 JMS suivants : LSF, NQE, DQS et PBS. Toutes les fonctionnalités citées dans le tableau sont relativement représentatives de l'ensemble des JMS existants.

- Tous ces JMS fonctionnent sur des *clusters* hétérogènes. Un nœud peut fonctionner sous n'importe lequel des systèmes Unix les plus courants. Cependant, ils ne supportent pas Windows NT, Windows 95, etc.
- Tous ces JMS ne nécessitent pas de matériel ou de logiciel supplémentaires pour fonctionner.
- Ils supportent aussi bien les applications de type parallèle que de type batch.
- Généralement, il y a un impact sur les exécutions des tâches locales à un nœud.
- Tous ces JMS utilisent des mécanismes de partage de charge pour utiliser le plus efficacement possible les ressources du *cluster*.
- Il n'y a pas de migration dynamique dans les JMS de la Figure 9.
- Un utilisateur ou l'administrateur peut suspendre et reprendre l'exécution d'une tâche.

Nom du JMS	Domaine commercial		Domaine public	
	LSF	NQE	DQS	PBS
Vendeur / créateur	Platform	Cray	FSU	NASA
Plate-forme (UNIX)	La plupart	La plupart	La plupart	Plusieurs
Type batch	Oui	Oui	Oui	Oui
Type application parallèle	Oui	Oui	Oui	Oui
Impact sur les tâches locales	Oui	Variable	Un peu	Variable
Partage de charges	Oui	Oui	Oui	Oui
Reprise	Oui	Non	Oui	Non
Migration	Non	Non	Non	Non
Etat de la tâche	Oui	Oui	Oui	Oui
Suspension et reprise de l'exécution	Oui	Oui	Oui	Oui
Ressources dynamiques	Oui	Oui	Oui	Oui
Interface Utilisateur	Interface graphique	Interface graphique, WEB	Interface graphique	Ligne de commande
Tolérance aux fautes	Oui	Quelques unes	Oui	Un peu
Sécurité	Kerberos	US DoD	Kerberos	Kerberos

Figure 9 : Vu d'ensemble des JMS

- La plupart des JMS ont à la fois une interface graphique et une interface utilisant une ligne de commande.
- La plupart des JMS utilisent le système d'authentification « Kerberos ».

I.3 L'exemple de NQE (Network Queuing Environnement)

Pour se faire une idée un petit peu plus précise de ce que sont les fonctionnalités d'un JMS, examinons le JMS Network Queuing Environnement (NQE) qui apparaît dans le tableau *Figure 9*. Lors des spécifications du JMS pour MPC, je me suis assez largement inspiré de NQE.

NQE a été développé par la compagnie *Cray* qui, par ailleurs, est connue par ses machines parallèles comme par exemple la **T3E**. NQE sert à exécuter des tâches sur un serveur où est installé le système NQS (*Network Queuing System*).

NQE s'utilise par l'intermédiaire d'une interface graphique ou d'une ligne de commande.

NQE est composé des éléments suivants :

- Une interface utilisateur
- Une interface administrateur qui permet de gérer le lancement des requêtes et les droits des différents utilisateurs
- Un système de récupération d'informations sur les différents nœuds et ce, en particulier, afin de faire du partage de charge
- Un système de gestion des tâches qui joue le rôle d'un *scheduler*
- Un système de transfert de fichiers qui utilise *FTP* (File Transfert Protocol) ou *RCP* (Remote CoPy) avec une procédure de reprise en cas d'échec du transfert

L'administrateur décide des droits des utilisateurs et en particulier certains utilisateurs peuvent avoir accès à un nombre restreint de services uniquement.

Les utilisateurs peuvent contrôler l'exécution de leurs requêtes et suivre l'état de leurs tâches comme par exemple les signaux reçus, les fichiers de log, la charge des nœuds où elles s'exécutent, etc.

NQE permet aux utilisateurs de :

- Emettre des requêtes pour lancer des tâches
- Router les requêtes automatiquement sans avoir besoin de connaître le nom des queues ou des serveurs
- Connaître l'état de la requête à intervalles de temps réguliers configurables par l'utilisateur
- Signaler le lancement des tâches
- Supprimer des requêtes des queues ou des tâches en cours d'exécution
- Connaître la charge des machines
- Reprendre l'exécution d'une tâche
- Transférer les log soit interactivement, soit par l'intermédiaire d'une requête
- Personnaliser l'environnement utilisateur
- Consulter le manuel online

L'administrateur utilise un fichier de configuration pour contrôler les utilisateurs. NQS utilise des fichiers de type *.rhosts* pour autoriser les connexions des utilisateurs.

Pour plus d'informations sur NQE, consulter <http://www.sgi.com/software/nqe/>.

II. Spécifications du JMS-MPC

Après l'étude des JMS existants, il est désormais plus facile de fournir des spécifications pour la machine MPC.

II.1 Pourquoi un JMS spécifique ?

Il est maintenant très clair qu'il est absolument nécessaire de disposer d'un logiciel ayant les fonctionnalités d'un JMS afin de pouvoir exploiter les performances de la machine MPC. En effet, le premier objectif est d'automatiser, et donc de rendre plus convivial, la mise en place et l'utilisation des services de la machine MPC et en particulier l'utilisation de son réseau haut débit HSL. Le deuxième objectif, fortement lié au premier, est de permettre à plusieurs utilisateurs de bénéficier simultanément, des services offerts par la machine.

Nous sommes en droit de nous demander pourquoi réaliser un JMS spécifique à la machine MPC alors qu'il en existe de nombreux sur le marché comme cela a été présenté à la section *Chapitre 2 : 1.2 Les systèmes existants*.

Tout d'abord, il est à noter que la machine MPC présente des caractéristiques très particulières. Le réseau HSL nécessite des procédés d'automatisation qui lui sont propres. Par ailleurs, pour l'instant, la seule façon d'utiliser ce réseau HSL est d'utiliser l'environnement de programmation PVM pour MPC. **Le JMS de la machine MPC doit d'une part, gérer l'administration du réseau HSL et d'autre part, permettre aux utilisateurs de lancer leurs applications PVM pour MPC.** Si on avait utilisé un JMS existant, il aurait fallu implémenter ces deux caractéristiques dans le JMS choisi.

Ensuite, la plupart des JMS existants possèdent eux aussi des fonctionnalités qui leur sont propres et qui seraient inutiles à l'administration de la machine MPC.

Enfin, les JMS existants étant assez complexes, il est sans aucun doute tout aussi facile de réaliser un JMS propre à la machine MPC que de partir d'un JMS existant et de l'adapter à la machine MPC.

II.2 Les règles de fonctionnement du JMS pour MPC

Il est à retenir du paragraphe précédant, que le JMS de la machine MPC a deux objectifs principaux qui sont :

- Gérer l'utilisation du réseau HSL et de PVM pour MPC ce qui signifie :
 - automatiser le chargement des drivers CMEM et HSL
 - automatiser le démarrage des démons *hslclient* et *hslserver*
 - automatiser le chargement du driver PVMDRIVER
 - automatiser le démarrage du démon PVM
- Permettre à plusieurs utilisateurs de tester leur application PVM sur la machine MPC ce qui se rapproche des fonctionnalités de tous les JMS

II.2.1 Principe général

La figure ci-dessous montre comment se décompose le JMS pour la machine MPC.

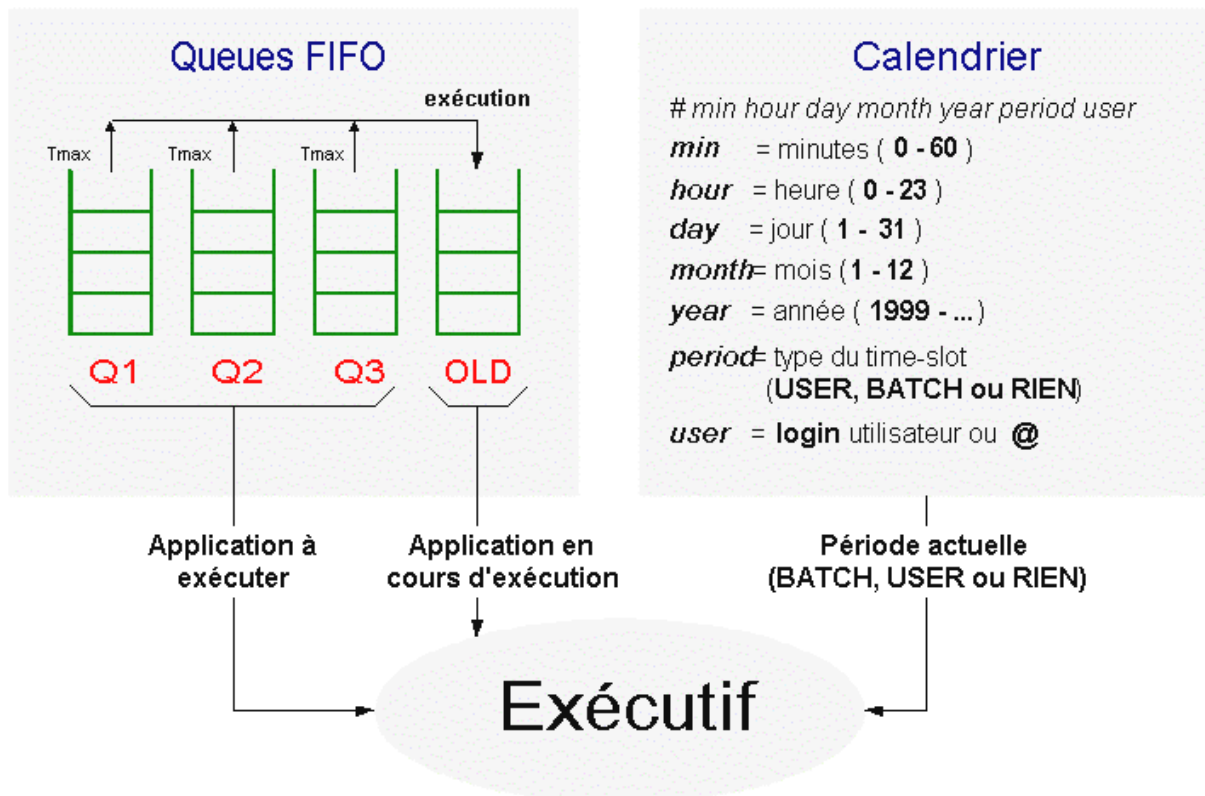


Figure 10 : Les composants du JMS pour MPC

Le JMS pour la machine MPC se décompose en trois principaux éléments :

- Les queues (ou files d'attente) qui permettent aux utilisateurs de lancer leurs applications PVM-MPC
- Le calendrier qui permet de définir différents types de périodes et éventuellement d'attribuer certaines périodes à un utilisateur.
- L'exécutif gère l'ensemble du JMS. En particulier, il applique les priorités définies à la fois par le calendrier et par la queue dans laquelle l'application à exécuter se trouve.

Le JMS a les caractéristiques suivantes :

- **Le JMS ne permet d'exécuter qu'une seule application PVM-MPC à la fois.** Ce choix est expliqué à la section [Chapitre 2 : II.4 Pourquoi ces choix ?](#)
- Le JMS est prévu pour fonctionner sur une architecture équivalente à celle de la machine MPC du LIP6. Cette architecture a été décrite à la section [Chapitre 1 : III. Description de la machine MPC.](#)
- Le nombre de nœuds de calcul est paramétrable.
- Chaque utilisateur doit posséder un compte UNIX non seulement sur la console mais aussi sur chacun des nœuds de calcul.
- Le JMS de la machine MPC est prévu pour fonctionner sous Unix FreeBSD. Cependant, le JMS peut fonctionner sur la plupart des plates-formes UNIX à condition d'effectuer quelques modifications.
- Les utilisateurs bénéficient de **deux interfaces** pour utiliser le JMS : une interface WEB et la ligne de commande.

II.2.2 Les queues

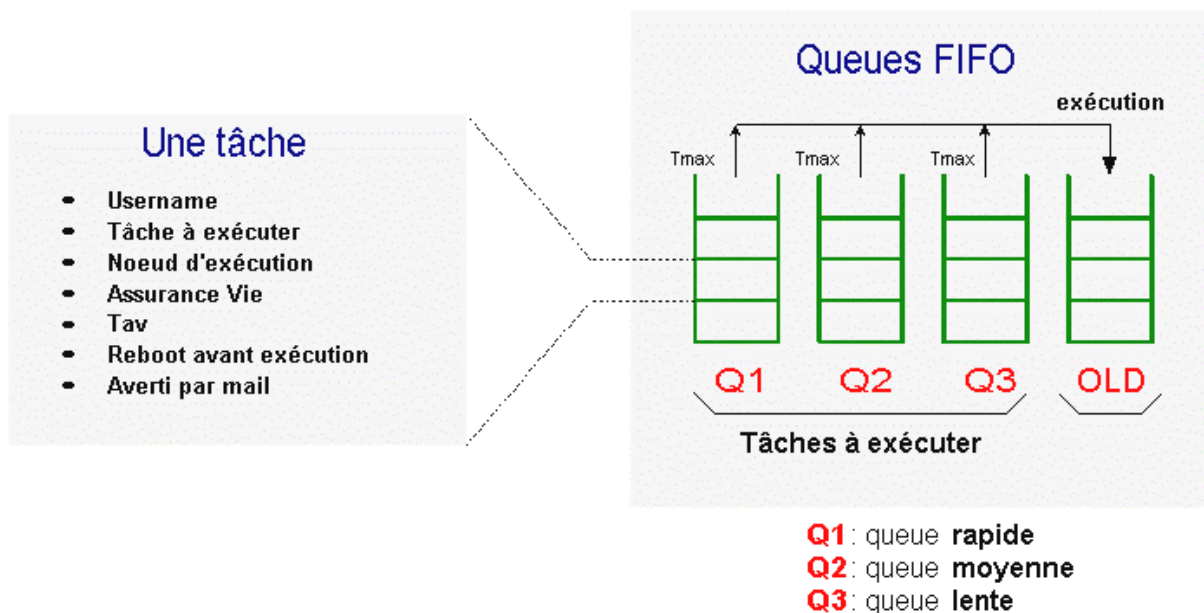


Figure 11 : Les files d'attente du JMS

La *Figure 11* ci-dessus présente les files d'attente qui compose le JMS. L'utilisateur a le choix entre 3 files lorsqu'il veut exécuter une application PVM-MPC :

- Une queue rapide **Q1**
- Une queue moyenne **Q2**
- Une queue lente **Q3**

Ces queues sont des files d'attente de type FIFO (*first-in, first-out*). Cela se traduit par une première règle de priorité :

P1 : si deux applications utilisateurs sont dans une même queue et ont des priorités égales par ailleurs, l'application qui a été mise en queue la première sera prioritaire sur l'autre.

Chacune de ces files d'attente a un paramètre *Tmax* fixé par l'administrateur. Tmax représente le temps (en minutes) d'exécution maximal des applications issues de la file. Il faut cependant respecter l'inégalité suivante :

$$T_{\max Q1} < T_{\max Q2} < T_{\max Q3}$$

On peut par exemple fixer $T_{\max Q1} = 10$ mns, $T_{\max Q2} = 60$ mns, $T_{\max Q3} = 600$ mns. C'est pour cela que Q1 est appelée *queue rapide*, Q3 *queue lente* ... Si le temps d'exécution d'une application issue d'une de ces 3 files dépasse le temps Tmax associée à cette file, l'application est tuée ainsi que tous les processus qu'elle a créés.

Cette classification des queues se traduit par une deuxième règle de priorité :

P2 : une application issue de la queue rapide est prioritaire sur une application issue de la queue moyenne, elle-même prioritaire sur une application issue de la queue lente. P2 est supérieure à P1.

Une dernière file compose le JMS : la queue *OLD*. Celle-ci réceptionne les applications dont l'exécution a été lancée. Elle permet de garder une trace des applications qui ont été exécutées.

Quand une application est mise en queue, les informations suivantes doivent être conservées :

- Le nom de l'utilisateur
- Le nom du binaire à exécuter
- Le numéro de nœud choisi pour l'exécution
- L'application a-t-elle une assurance vie ? (cf. *Chapitre 2 : II.3 Les fonctionnalités*)
- Le temps *Tav* associé à une éventuelle assurance vie.
- L'utilisateur souhaite-t-il rebooter les nœuds de calcul avant que son application soit exécutée ?
- L'utilisateur souhaite-t-il être informé par mail du devenir de son application ?

II.2.3 Le calendrier

Un deuxième élément important compose le JMS de la machine MPC. Il s'agit du calendrier.

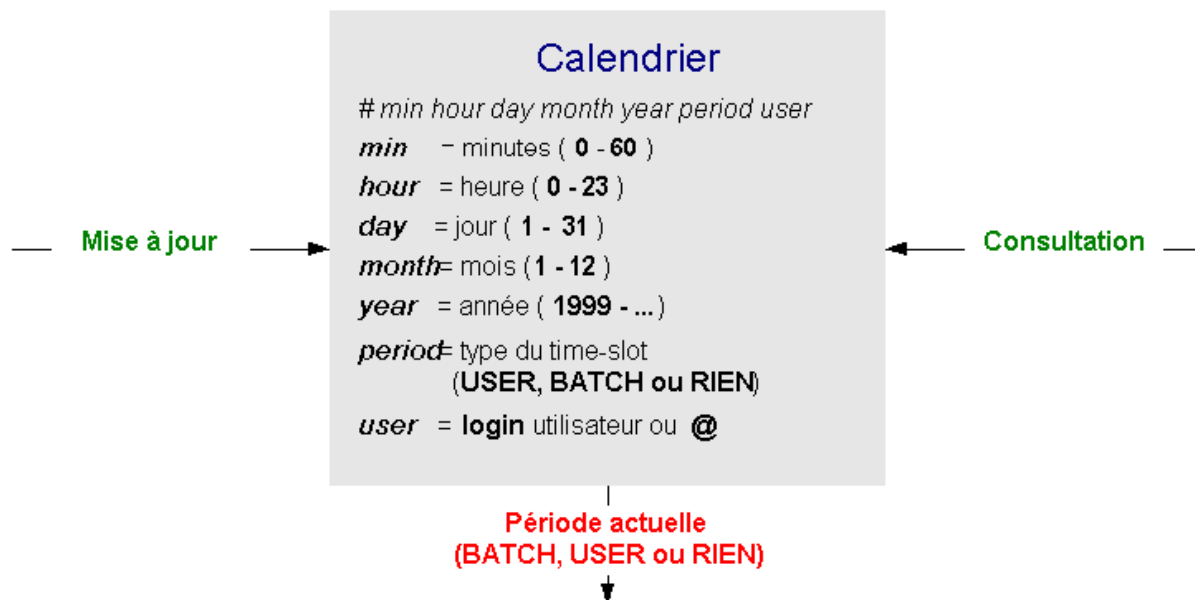


Figure 12 : Le calendrier

Le calendrier sert à définir des périodes. L'administrateur a la charge de la gestion de ce calendrier. Il y a trois types de périodes possibles :

- Période de type **BATCH** : c'est la période utilisée en temps normal. Tous les utilisateurs sont équivalents. Les deux seules règles de priorité sont P1 et P2 énoncées à la section *Chapitre 2 : II.2.2 Les queues*.
- Période de type **USER** (ou utilisateur) : ce type de période permet de privilégier un utilisateur par rapport à tous les autres en donnant son *login*. Si une application a été lancée dans une période BATCH et que la période a été ensuite attribuée à un utilisateur, cette application peut-être tuée avant la fin de son exécution si l'utilisateur en question veut lui-même lancer une application. Une nouvelle règle de priorité intervient alors :

P3 : Si la période est réservée à un utilisateur (type USER) particulier alors ses applications sont prioritaires sur toutes les autres. P3 est supérieure à P1 et P2.

- Période de type **RIEN** : ce type de période permet à l'administrateur de « geler » l'exécution des applications. Si une application est en cours d'exécution et que le type de la période devient RIEN alors l'application est tuée.

Le calendrier peut être consulté par tous les utilisateurs et doit être maintenu et mis à jour par l'administrateur.

II.2.4 L'exécutif

L'exécutif est l'organe central du JMS pour la machine MPC.

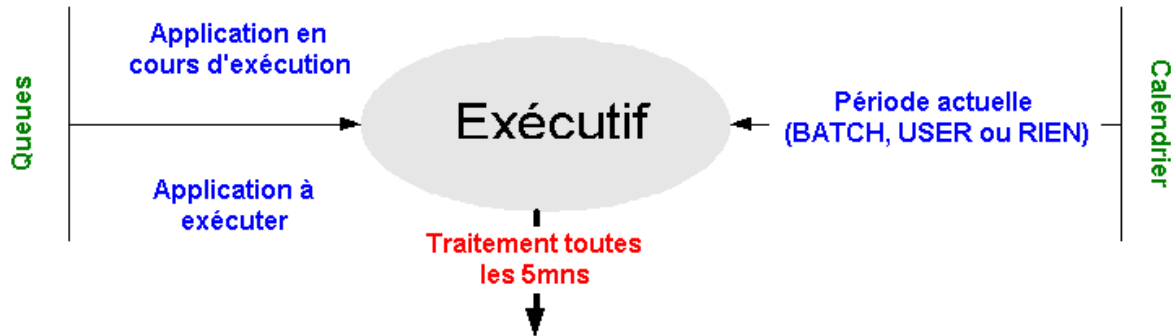


Figure 13 : L'exécutif

Il gère l'ensemble du JMS et s'occupe de lancer les applications que les utilisateurs stockent dans les queues.

L'exécutif lit le calendrier pour savoir dans quel type de période il se trouve. Il lit ensuite le contenu des queues Q1, Q2, Q3 et, suivant le type de période, classe les applications contenues dans ces queues par ordre de priorité.

Par ailleurs, il lit le contenu de la file OLD pour connaître la dernière application à avoir été exécutée. Il regarde si celle-ci est effectivement en cours d'exécution. Si c'est le cas, il détermine alors si cette application doit être tuée ou non.

Enfin, il détermine si une application doit être lancée et, si c'est le cas, lance cette application sur le nœud 0. En effet, une application PVM-MPC se lance toujours sur le nœud 0 (*cf. Chapitre 1 : III.3.1 Architecture générale de PVM-MPC*). Avant chaque exécution, les démons PVM et MPC sont tués puis relancés.

L'exécutif est lancé toutes les 5 minutes afin de suivre l'évolution de l'exécution des applications.

L'algorithme de l'exécutif est présenté à la section *Chapitre 2 : III.3 L'exécutif (Figure 17)*.

II.3 Les fonctionnalités

Les fonctionnalités du JMS peuvent se décomposer en 2 parties : les fonctionnalités utilisateur et les fonctionnalités administrateur.

II.3.1 L'utilisateur

Un simple utilisateur du JMS dispose des fonctionnalités suivantes :

- Mettre une application dans la queue de son choix
- Consulter l'état des files d'attente
- Supprimer une application d'une queue (si elle lui appartient)
- Consulter l'application en cours d'exécution
- Tuer l'application en cours d'exécution (si elle lui appartient)
- Consulter l'état du calendrier
- Devenir l'utilisateur privilégié (cf. *Les périodes de type USER sans login*)
- Tester l'application PVM-MPC en cours d'exécution
- Consulter la liste des drivers et démons MPC et PVM sur tous les nœuds de calcul
- Consulter la liste de tous les processus (de l'utilisateur) sur tous les nœuds
- Rebooter les nœuds de calcul avant l'exécution de son application (si cela est permis par l'administrateur)
- Rapatrier les fichiers de log

Les périodes de type USER sans login

L'administrateur du JMS peut, lorsqu'il définit les périodes du calendrier, décider de réserver des périodes de type USER sans pourtant donner le login d'un utilisateur privilégié. Ainsi, durant ce type de période, si un utilisateur éprouve le besoin de devenir l'utilisateur privilégié, il le peut. Tant qu'aucun utilisateur n'a décidé de s'attribuer la période, le fonctionnement du JMS est équivalent au type BATCH.

L'assurance vie

L'utilisateur peut décider de couvrir son application par une assurance vie quand il la met dans une file. Qu'est-ce que cela signifie ? Cela permet d'empêcher que l'application ne soit lancée s'il y a des risques qu'elle soit tuée. Cela peut se produire si le type de période change avant la fin de l'exécution de l'application. Pour bénéficier de l'assurance vie, l'utilisateur indique un paramètre : **Tav** en minutes. Tav représente le temps minimum d'exécution de l'application (pendant Tav, l'application ne sera pas tuée). En revanche, si l'utilisateur prend une assurance vie, il se peut que l'exécution de son application soit retardée.

Les mails

L'utilisateur peut être averti par e-mail (s'il le désire uniquement) des événements suivants :

- L'exécution de son application va commencer
- Son application est tuée par le JMS
- Son application est testée par le JMS

II.3.2 L'administrateur

L'administrateur du JMS dispose des fonctionnalités suivantes :

- Configurer le calendrier
- Configurer les files d'attente
- Arrêter / Redémarrer l'exécutif
- Vider toutes les files d'attente
- Rebooter les nœuds de calcul
- Devenir un des utilisateurs du JMS

Par ailleurs, l'administrateur doit gérer l'ensemble des utilisateurs de la machine MPC et en particulier leur permettre de se connecter à la machine, etc.

II.4 Pourquoi ces choix ?

Nous pouvons maintenant nous interroger sur les choix qui ont été faits pour le JMS de la machine MPC.

L'architecture (une console + les nœuds de calcul)

Le principe de fonctionnement du JMS nécessite d'avoir une console en amont des nœuds de calcul. D'ailleurs, le cœur même du JMS se trouve sur la console. Cette restriction quant à l'architecture est surtout due au fait que les nœuds de calcul peuvent être rebooter. Or, il n'est pas envisageable de faire tourner le JMS sur un nœud pouvant être rebooter. Les utilisateurs peuvent se connecter aux nœuds de calcul mais leur connexion peut être coupée à tout moment. En revanche, une connexion sur la console est beaucoup plus viable. Par ailleurs, la console sert essentiellement à administrer l'ensemble des nœuds de calcul donc il est opportun de faire tourner le JMS sur la console de la machine MPC.

Le JMS est mono-exécution (*Dedicated mode*)

La philosophie de la machine MPC est de permettre à des utilisateurs de bénéficier d'une puissance de calcul gigantesque. Ils peuvent ainsi faire tourner de grosses applications en espérant avoir un gain de performances en comparaison à d'autres machines parallèles. Si l'on suit cette philosophie, il serait dénué de sens de faire tourner un grand nombre d'applications en même temps qui se trouveraient alors en compétition. C'est pourquoi, ce mode d'utilisation a été choisi.

Le reboot

Il a été jugé utile de permettre le redémarrage total des nœuds de calcul. En effet, la machine MPC n'est pas encore d'une stabilité suffisante et il peut arriver que des drivers (MPC ou PVM) ne soient plus correctement chargés après l'exécution de plusieurs applications PVM-MPC. Il est alors nécessaire de redémarrer les nœuds de calcul afin d'être certain que ces pilotes soient correctement chargés. En particulier, avant l'exécution d'une grosse application, il est sans doute préférable de perdre 3 minutes pour rebooter l'ensemble des nœuds de calcul. Cependant, l'administrateur peut empêcher le redémarrage des nœuds s'il le souhaite.

Relancer les démons avant chaque exécution

Il est préférable de tuer et relancer les démons PVM et MPC entre chaque exécution, et ce, pour être sûr qu'ils soient bien lancés. C'est pourquoi, le JMS de la machine MPC fait ces opérations automatiquement.

Les deux interfaces

Il est de toute façon nécessaire d'avoir une interface utilisateur utilisant la ligne de commande. Cependant, il a été jugé utile de fournir une autre interface plus conviviale. Une interface *CGI* (interface WEB) a été choisie. Tout d'abord, elle est relativement facile à programmer comparativement à une interface XWINDOW par exemple. Ensuite, le but est également de permettre à des utilisateurs extérieurs de se connecter sur la machine MPC. Une interface WEB est un moyen souvent utilisé pour permettre ce type de connexion.

Les files d'attente

Elles sont au nombre de trois. En effet, il est nécessaire de classer les applications par leur temps d'exécution à priori, le but étant de privilégier les applications dont le temps d'exécution n'est pas trop important. C'est pour cela que les utilisateurs disposent d'une queue dite rapide, d'une queue moyenne et d'une queue lente.

Le calendrier

Le but du calendrier est de permettre de définir différentes périodes d'utilisation du JMS. En effet, à un moment donné, il est par exemple souhaitable que beaucoup d'utilisateurs puissent exécuter des applications dont l'exécution est relativement courte. Dans ce cas, il suffit de se placer dans une période de type BATCH. A un autre moment, il sera peut-être préférable de réserver la machine à un utilisateur donné qui veut exécuter des applications importantes. Il faut alors se placer dans une période de type USER. Enfin, il est souhaitable pour l'administrateur de pouvoir stopper l'exécution des applications sans pour autant arrêter le JMS (les utilisateurs peuvent continuer à mettre leurs applications dans les queues...). Cela peut, par exemple, être utile lorsque l'administrateur fait de la maintenance sur les nœuds de calcul. Il suffit pour cela de se placer dans une période de type RIEN.

L'administrateur peut devenir un simple utilisateur

Cette fonctionnalité permet à l'administrateur de « se mettre dans la peau » de n'importe quel utilisateur du JMS. Cela est toujours utile au cas où des utilisateurs aient des problèmes.

Pas de partage de charge, de prise en compte des ressources et de migration

Toutes ces fonctions n'ont pas été implémentées dans le JMS de la machine MPC pour plusieurs raisons. Tout d'abord, les processus de migration et de partage de charge sont très complexes et il aurait été difficile de les mettre en œuvre sur la machine MPC. Par ailleurs, étant donné le fonctionnement en mode mono-exécution (*Dedicated mode*), l'application qui s'exécute dispose à elle seule de l'ensemble des ressources de la machine. Il est donc inutile de prendre en compte les ressources demandées par l'application dans les règles de priorités. Enfin, étant donné que la machine est totalement dédié à l'application qui s'exécute, l'utilisateur peut s'arranger pour que le partage de charge soit correct.

III. Réalisation du JMS-MPC

Nous pouvons maintenant examiner la façon dont le JMS de la machine MPC a été réalisé.

III.1 Architecture logicielle

Le JMS de la machine MPC est composé de différents types de fichiers qui sont regroupés dans un seul répertoire : **MPC-JMS**.

III.1.1 Le répertoire MPC-JMS

Ce répertoire se trouve sur la console de la machine MPC. Il contient 7 sous-répertoires (cf. *Figure 14*).

- **admin-bin** : répertoire contenant tous les shells UNIX relatifs au chargement / déchargement des drivers MPC et PVM, ainsi qu'au lancement et à l'arrêt des démons MPC et PVM
 - **nodes-bin** : ensemble des fichiers devant être sur chaque nœud de calcul afin de lancer les démons et drivers de la machine MPC (en particulier, ce répertoire doit contenir les fichiers de tables de routage du réseau HSL ainsi que les fichiers de configuration du réseau)
 - **logs** : répertoire contenant les fichiers de log correspondant à l'exécution de certains shells de *admin-bin*
- **bin** : répertoire contenant des liens statiques sur tous les binaires nécessaires aux utilisateurs du JMS (ce répertoire doit figurer dans la variable d'environnement PATH)
- **conf** : ensemble des fichiers de configuration du JMS
 - **mails** : texte des mails envoyés aux utilisateurs par le JMS
- **jms-bin** : ensemble des binaires et des shells utilisés par l'interface ligne de commande du JMS

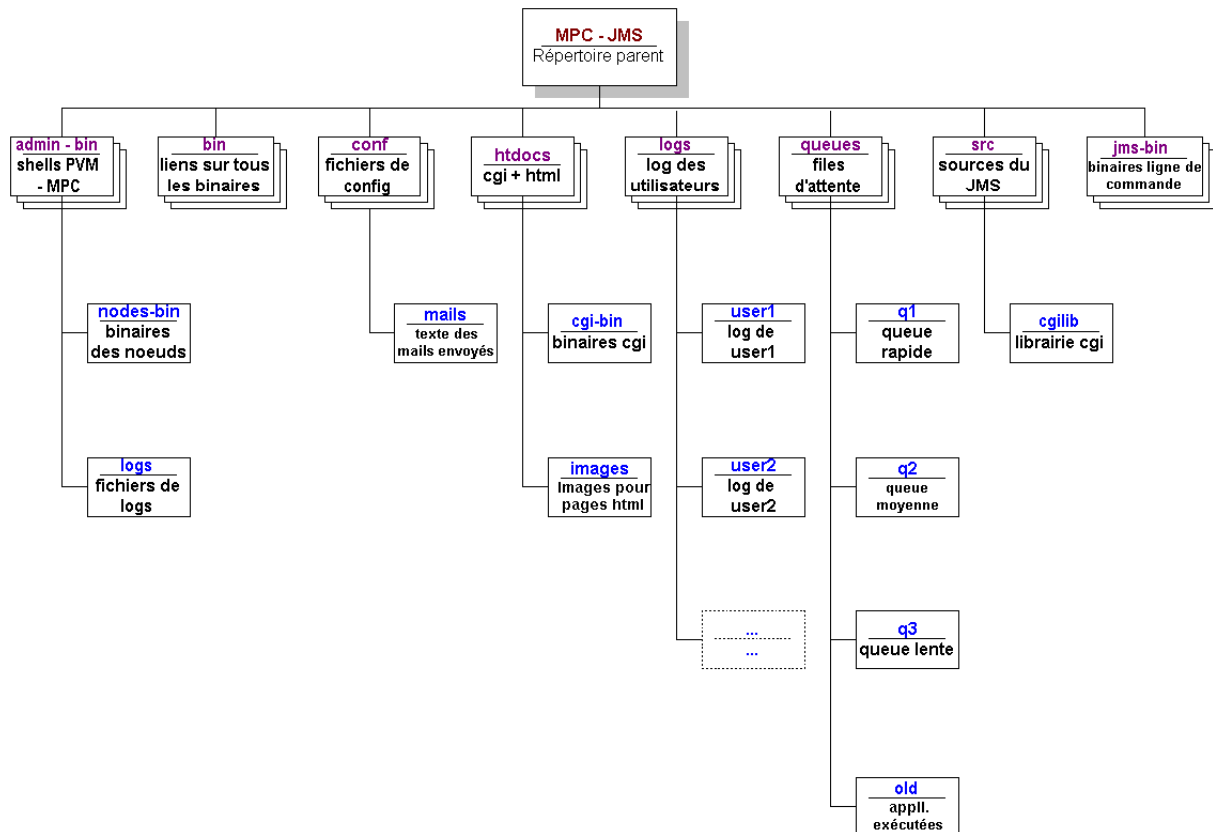


Figure 14 : Arborescence du JMS-MPC

- **htdocs** : ensemble des fichiers relatifs à l'interface WWW (fichiers hypertext, exécutables cgi, etc.)
 - **cgi-bin** : répertoire contenant tous les exécutables cgi (utilisés uniquement par l'interface WWW)
 - **images** : ensemble des images utilisées dans les fichiers hypertext du JMS
- **logs** : ensemble des fichiers de log des utilisateurs. Chaque utilisateur a son propre sous répertoire. Lorsqu'un utilisateur transfère ses fichiers de log présents sur tous les nœuds de la machine MPC, tous ces fichiers sont rapatriés dans *logs/user/node*i** où *i* est le numéro de nœud
- **queues** : ensemble des fichiers utilisateur contenus dans les files d'attente du JMS, un fichier correspondant à une application utilisateur
 - **q1** : fichiers relatifs aux applications de la queue rapide
 - **q2** : fichiers relatifs aux applications de la queue moyenne
 - **q3** : fichiers relatifs aux applications de la queue lente
 - **old** : fichiers provenant de q1, q2 ou q3 relatifs aux applications dont l'exécution a été lancée
- **src** : ensemble des sources du JMS pour la machine MPC
 - **cgilib** : source de la bibliothèque cgi utilisée par le JMS

III.1.2 Les différents types de fichiers

Le JMS pour la machine MPC est constitué de :

- **25 shells UNIX** : les shells d'administration de la machine et les shells utilisés par l'exécutif ou par l'interface cgi (par exemple, pour lancer ou tuer les applications utilisateur, etc.)
- **10 fichiers hypertext (HTML)** : ces fichiers sont utilisés par l'interface WWW et les exécutables cgi
- **11 binaires (écrits en langage C)** : les utilisateurs souhaitant utiliser le JMS avec l'interface « ligne de commande » se servent de ces exécutables
- **27 exécutables cgi (écrits en langage C et CGI)** : exécutables utilisables uniquement via l'interface WWW
- **23 liens statiques** sur les binaires devant figurer dans la variable d'environnement PATH des utilisateurs
- **5 fichiers de configuration** : (cf. *Chapitre 2 : III.5 Les fichiers de configuration*)
 - *config* : fichier de configuration générale
 - *calendar* : fichier calendrier contenant les périodes utilisateur
 - *config_queues* : fichier de configuration des files d'attente
 - *beuser.login* : fichier contenant le login utilisateur choisi par l'administrateur
 - *lock* : fichier vide permettant de réaliser un mécanisme de *lock* afin d'éviter les conflits logiciels (cf. *Chapitre 2 : III.6.3 Les locks*)
- **Des documentations** (guide d'utilisation et manuel d'installation), un **README**
- **Les sources** (cf. *Figure 15*)

Les sources peuvent se décomposer en deux parties : les sources relatifs à l'interface « ligne de commande » et ceux relatifs à l'interface CGI (cf. tableau ci-dessous).

Ligne de commande		CGI	
	conf, queue, calendrier		conf, queue, calendrier, cgi
3 librairies	libconf.a libqueue.a libcalendrier.a	4 librairies	libconfcgi.a libqueuecgi.a libcalendriercgi.a libcgic.a
14 fichiers sources en langage C		29 fichiers sources en langage CGI-C	
Makefile : <i>make binex</i>		Makefile : <i>make cgis</i>	

Figure 15 : Les sources du JMS

Les librairies *libconf.a* et *libconcgi.a* contiennent toutes les fonctions et procédures de configuration générale du JMS (procédure de lock, fonctions retournant des informations de configuration, etc.).

Les bibliothèques *libqueue.a* et *libqueuecgi.a* contiennent toutes les fonctions et procédures relatives aux traitements des files d'attente (procédures de tri, fonctions retournant l'état des queues, procédures liées à la configuration des queues).

Les bibliothèques *libcalendrier.a* et *libcalendriercgi.a* contiennent toutes les fonctions et procédures relatives aux traitements du calendrier (recherche de la période actuelle, mise à jour, assurance vie).

III.2 Architecture des files d'attente

Les files d'attente sont au nombre de 4 (cf. *Figure 11 : Les files d'attente du JMS*). D'une manière générale, les files d'attente sont représentées par des répertoires. C'est la solution simple que nous avons adoptée.

Chaque fichier d'un de ces répertoires représente une application utilisateur. Il contient toutes les informations nécessaires à l'exécution de cette application par l'exécutif du JMS. Voici un exemple type de fichier représentant l'application PVM-MPC *hello*.



Figure 16 : Élément d'une file d'attente

L'utilisateur de login *olivier* veut lancer l'application PVM-MPC *hello* (c'est à dire lancer la tâche maître PVM-MPC dont le nom de l'exécutable est *hello*) sur le **nœud 0** (la tâche maître se lance toujours sur le nœud 0 qui est le nœud du démon, cf. *III.3.1 Architecture générale de PVM-MPC*). Il a choisi de prendre une **assurance vie** avec comme paramètre **20 minutes** ce qui signifie que cette application ne sera lancée que si elle est assurée de pouvoir s'exécuter au moins 20 minutes sans être tuée. Par conséquent, l'utilisateur doit placer cette application dans une file d'attente dont le temps d'exécution maximal T_{max} est supérieur à 20 minutes. L'utilisateur ne souhaite pas **redémarrer** les nœuds de calcul avant l'exécution mais souhaite être averti par **mail** du devenir de son application.

Le nom donné au fichier représentant une application est construit de la manière suivante : le nom de l'utilisateur, un trait d'union, l'année sur 4 chiffres, le mois sur 2 chiffres, le jour sur 2 chiffres, un trait d'union, l'heure sur 2 chiffres, les minutes sur 2 chiffres et les secondes sur 2 chiffres. Dans l'exemple de la figure 16, le nom du fichier est **olivier-19990622-150208**. Cela signifie que l'utilisateur olivier a mis cette application dans une queue le 22 juin 1999 à 15h 2mns et 8s. Cette façon de nommer les fichiers permet d'être sûr que deux applications différentes ne correspondent pas au même nom de fichier.

Un utilisateur ne peut insérer des fichiers que dans les queues *q1*, *q2* et *q3*. Lorsqu'un fichier est migré dans la queue **old** par l'exécutif car son exécution vient d'être lancée, le suffixe **.q1** ou **.q2** ou **.q3** est ajouté au nom de fichier afin de distinguer la provenance du fichier.

Les fichiers de toutes les files d'attente sont classés par la date de création du fichier suivant l'algorithme de tri rapide présenté à la section [Chapitre 2 : III.6.2 L'algorithme de tri](#).

Les files d'attente peuvent être vidées par l'administrateur. En particulier, la queue *old* doit être vidée de temps en temps. Le vidage de cette file consiste en la suppression de tous les fichiers, excepté le plus récent. En effet, si une application est en cours d'exécution, il ne faut surtout pas effacer le fichier correspondant qui se trouve dans la file *old*.

III.3 L'exécutif

L'exécutif décide de la tâche qui doit être exécutée à un moment donné. Pour cela, il va lire dans le calendrier la période dans laquelle le JMS se trouve (notée TS dans l'algorithme ci-dessous). Il va également extraire de la queue *old* le fichier le plus récent qui est noté AC (application en cours) dans l'algorithme. Cependant, AC n'a de signification que si l'application concernée est effectivement en cours d'exécution. Pour cela, l'exécutif va inspecter les nœuds de calcul afin de voir si AC est toujours en cours d'exécution. Dans ce cas, AC = 1 sinon AC = 0. Enfin, l'exécutif parcourt les répertoires *q1*, *q2*, et *q3* pour classer les applications suivant les priorités P1, P2, P3 énoncées à la section [Chapitre 2 : II.2 Les règles de fonctionnement du JMS pour MPC](#). Cela correspond aux procédures *Remplir_user* et *Remplir_batch* dans l'algorithme. Le classement des applications n'est pas le même si un utilisateur est privilégié ou non.

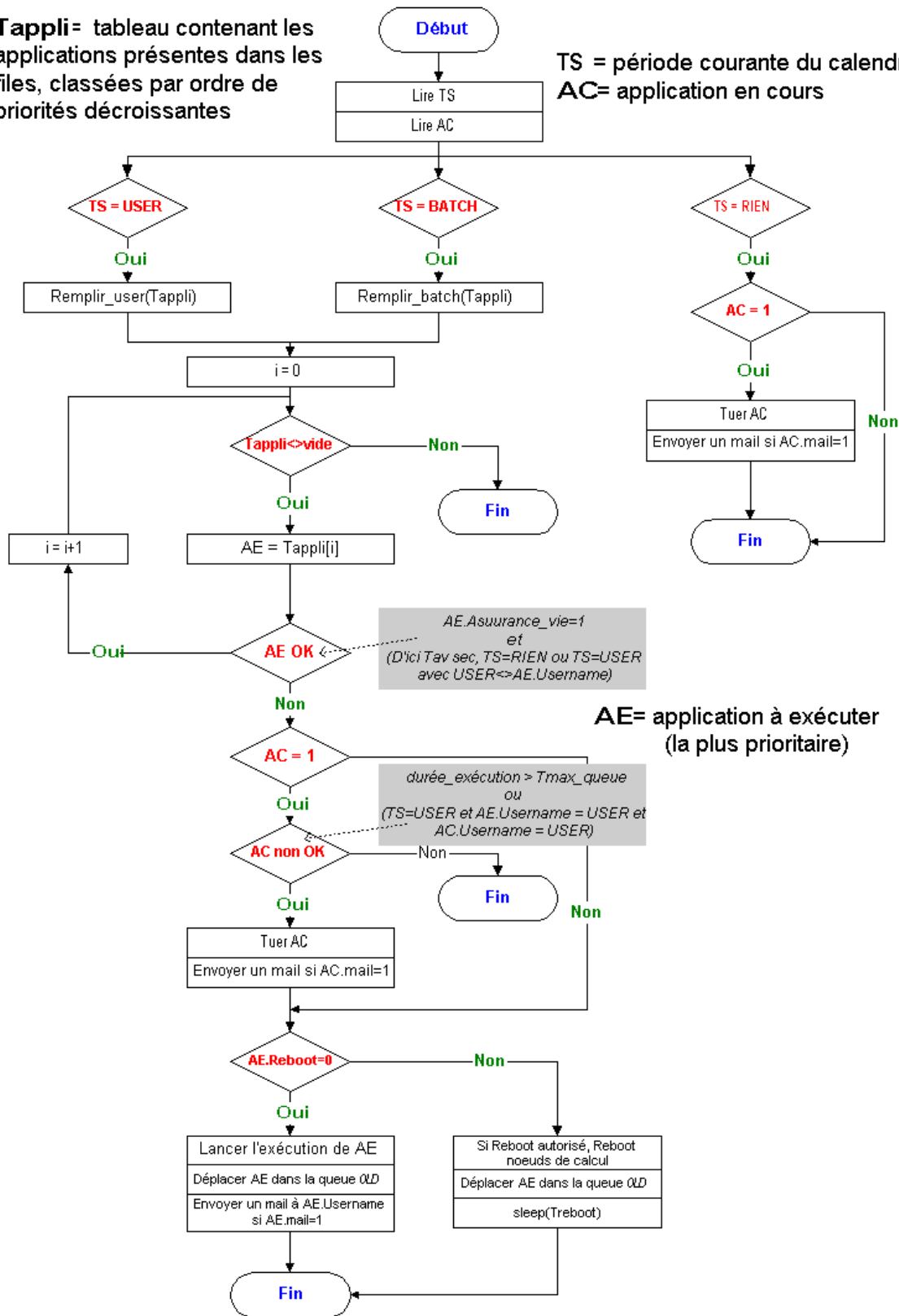
Pour lancer les tâches sur les nœuds distants, l'exécutif utilise **ssh** (secured shell). Une fenêtre *xterm* est alors ouverte pour suivre l'exécution de l'application.

L'exécutif est lancé, toutes les 5 minutes par exemple, par la **crontab** de la console. Cependant, l'administrateur peut décider d'arrêter ou de reprendre l'exécution de l'exécutif. Pour cela, le mécanisme est le suivant : en réalité, la crontab ne lance pas l'exécutif directement. Elle exécute un shell UNIX qui ne lance l'exécutif qu'à la condition que le fichier nommé « *gojms* » (ce fichier est vide) soit dans le répertoire *conf* du JMS. Pour arrêter ou redémarrer l'exécutif, il suffit donc d'effacer ou de créer le fichier *gojms*.

Ci-dessous, l'algorithme de l'exécutif est détaillé.

Tappli= tableau contenant les applications présentes dans les files, classées par ordre de priorités décroissantes

TS = période courante du calendrier
AC= application en cours



AE= application à exécuter (la plus prioritaire)

Figure 17 : Algorithme de l'exécutif

III.4 Les utilisateurs

Le JMS permet de se connecter à la machine MPC pour lancer des applications PVM-MPC. Cela peut se faire par exemple, par l'intermédiaire d'Internet. Seulement, pour avoir accès à ces services, il faut être déclaré en tant qu'utilisateur du JMS. Comment devenir un utilisateur du JMS ?

La première des choses à faire est de créer un compte UNIX à l'utilisateur non seulement sur la console mais aussi sur l'ensemble des nœuds de calcul. Sur les nœuds de calcul, l'utilisateur doit avoir un répertoire MPC-JMS dans lequel figureront les exécutable PVM-MPC qu'il souhaite exécuter et les fichiers de log éventuels créés lors des exécutions. Sur la console, il aura accès à toutes les commandes du JMS fournies par l'interface « ligne de commande ».

Ensuite, si l'utilisateur veut bénéficier de l'interface CGI, il faut le déclarer dans le fichier *htpasswd.users* du répertoire *htdocs* en utilisant la commande *htpasswd*.

L'administrateur doit s'occuper de la gestion des utilisateurs.

III.5 Les fichiers de configuration

Il y a principalement 5 fichiers de configuration :

- ***config*** : fichier de configuration générale
- ***calendar*** : fichier calendrier contenant les périodes utilisateur
- ***config_queues*** : fichier de configuration des files d'attente
- ***beuser.login*** : fichier contenant le login utilisateur choisi par l'administrateur
- ***lock*** : fichier vide permettant de réaliser un mécanisme de *lock* afin d'éviter les conflits logiciels (cf. [Chapitre 2 : III.6.3 Les locks](#))

L'administrateur ne doit jamais modifier les fichiers *beuser.login* et *lock*. Ces deux fichiers sont entièrement gérés par le JMS lui-même.

III.5.1 Le fichier de configuration générale (*config*)

Ce fichier doit être modifié principalement lors de l'installation du JMS (cf. [Annexe 3](#)). Il contient toutes les informations nécessaires au fonctionnement du JMS comme les chemins d'accès aux répertoires, les informations sur les nœuds de calcul (nombre, noms), la configuration du reboot des nœuds, et les informations nécessaires à l'utilisation du démon *httpd* (cf. [Figure 18](#)).

```

# Configuration file
# NOES
# Be carefull : the node 0 is the node where the PVM daemon is launched
NUMBERNODES = 4
NODE0 = mpc1-lip6.lip6.fr
NODE1 = mpc2-lip6.lip6.fr
NODE2 = mpc3-lip6.lip6.fr
NODE3 = mpc4-lip6.lip6.fr

# DIR
# Every directory path is relative to JMSDIR
JMSDIR = /usr/local/MPC-JMS
QUEUESDIR = queues
CONFDIR = conf
BINDIR = bin
SRCDIR = src
HTMLEDIR = htdocs
CGIBINDIR = htdocs/cgibin
MPCDIR = /usr/local/home/alex/MPC-OS-LIP6
LOGDIR = /usr/local/MPC-JMS/logs
# path relative to a user directory on calcul nodes
LOGNODESDIR = MPC-JMS/logs

# WEB
# Here specify the URL for accessing .html of JMS
URL = http://mpc0-lip6.lip6.fr
# Here specify the login used by httpd
LOGIN_HTTPD = wwwcgl

# Reboot of all calcul-nodes before each execution ?
# Indicate time of reboot in minutes
REBOOT = 1
TREBOOT = 3

```

Fichier de configuration générale config

Informations relatives aux noeuds de calcul

Chemins de tous les répertoires utilisés

Informations relatives à l'interface cgi

Informations concernant le reboot des noeuds

Figure 18 : Le fichier config

Ce fichier est lu par tous les shells et par tous les exécutable C ou CGI du JMS à chaque exécution.

LOGNODESDIR est le répertoire local aux nœuds de calcul où les utilisateurs mettent les fichiers de log de leur applications.

URL est le chemin d'accès via le WEB au répertoire *htdocs* du JMS.

LOGIN_HTTPD est le login du démon http. Tous les exécutable cgi seront exécutés par ce pseudo-utilisateur.

L'administrateur peut par exemple interdire le reboot des nœuds de calcul en mettant REBOOT = 0. TREBOOT est le temps approximatif de redémarrage des nœuds en minutes.

III.5.2 Le fichier de configuration des queues (*config_queues*)

Ce fichier peut être modifié n'importe quand par l'administrateur. Cependant, il est préférable que celui-ci configure les files d'attente en utilisant l'exécutable *configq* du JMS ou bien l'interface WWW où cette fonctionnalité est disponible. Alors, le fichier *config_queues* sera modifié automatiquement.

Ce fichier de configuration contient le paramètre *Tmax* pour chaque file d'attente. Les explications concernant *Tmax* sont données à la section [Chapitre 2 : II.2.2 Les queues](#).

Voici un exemple de configuration :

```

Fichier config_queues
# Tmax (in mns) for q1 (fast queue)
10
# Tmax (in mns) for q2 (middle queue)
60
# Tmax (in mns) for q3 (slow queue)
360

```

Figure 19 : Le fichier de configuration des queues

Ce fichier est utilisé par toutes les fonctions et procédures qui gèrent les files d'attente.

III.5.3 Le fichier de configuration du calendrier (*calendar*)

Ce fichier doit être mis à jour régulièrement par l'administrateur. Il sert à définir les périodes utilisateur. Chaque ligne représente un changement de période.

Calendrier

min hour day month year period user
min = minutes (**0 - 59**)
hour = heure (**0 - 23**)
day = jour (**1 - 31**)
month = mois (**1 - 12**)
year = année (**1999 - ...**)
period = type du time-slot
 (**USER, BATCH ou RIEN**)
user = **login** utilisateur ou **@**

```

Fichier calendar
23 4 3 6 1999 USER olivier
22 5 10 6 1999 USER @
15 3 5 4 2000 BATCH @
03/06/1999, 04h23
10/06/1999, 05h22
05/04/2000, 03h15

```

Figure 20 : Le fichier calendrier

A la date indiquée sur une ligne, il lui correspond une période et un utilisateur. Si aucun utilisateur n'est attribué, il faut mettre le symbole @. A chaque fois que le calendrier est consulté, les lignes sont triées par dates croissantes et les lignes dont la date est dépassée sont effacées exceptée la plus récente car elle contient les informations de la période actuelle. Dans l'exemple de la figure 20, si l'on est le 12 juin 1999, la période actuelle est de type *USER* avec l'utilisateur non renseigné ce qui signifie que le premier utilisateur qui le désire peut réserver la période entière.

Ce fichier est utilisé par toutes les fonctions et procédures qui gèrent le calendrier.

III.6 Quelques points techniques

III.6.1 Les bibliothèques

Trois bibliothèques de fonctions et procédures ont été conçues pour le JMS (cf. *Figure 15 : Les sources du JMS*).

La bibliothèque *conf*

La première bibliothèque appelée *conf* contient toutes les fonctions et procédures utilisées par tous les exécutables comme par exemple la fonction qui lit le fichier de configuration *config*.

La définition C de la structure utilisée par cette bibliothèque est :

```
/* Struct of config */
typedef struct config
{
    int nbnodes ;
    char jmsdir[MAX_PATH] ;
    char queuesdir[MAX_PATH] ;
    char confdir[MAX_PATH] ;
    char bindir[MAX_PATH] ;
    char srcdir[MAX_PATH] ;
    char htmldir[MAX_PATH] ;
    char cgibindir[MAX_PATH] ;
    char jmsurl[MAX_URL] ;
    char login_httpd[MAX_USERNAME] ;
    int reb ;
    int treb ;
} config ;
```

La bibliothèque *calendrier*

Cette bibliothèque contient toutes les fonctions et procédures qui permettent de traiter le fichier calendrier (cf. *Figure 20 : Le fichier calendrier*).

La définition en langage C de la structure définie dans cette librairie est :

```
/* Struct of calendar : 1 record = 1 line in the file calendar */
typedef struct line
{
    int min ;
    int hour ;
    int day ;
    int month ;
    int year ;
    char ts[MAX_TS] ;
    char username[MAX_USERNAME] ;
} line ;
```

Les définitions en langage C des principales fonctions et procédures de cette librairie sont :

```
/* Return the number of lines of the file */
int nb_lines(char *) ;

/* Fill in calend with lines of the file "calendrier" */
void file_in_struct(char *, line *calend) ;

/* Fill in the file "calendrier" with lines of calend */
void struct_in_file(line *calend, char *) ;

/* Compare lines i and j in function of date
   Return 1 if date[i] >= date[j] */
int compare_line(line *, int i, int j) ;

/* Exchange line i with line j */
void exchange_line(line *, int i, int j) ;

/* Delete line i of calend */
void del_line(line *calend, int i) ;

/* Sort calend[left]..calend[right] by increasing date - Algorithm of Quick
Sort */
void sort(line *calend, int left, int right) ;

/* Delete all lines with an old date except the most recent one */
void update(line *) ;

/* Return 1 if in [date,date+tmin] : TS=RIEN or TS=USER with USER=aeuser
 * else Return 0 */
int assurance_vie(int tmin, char *aeuser, char *filename) ;
```

La librairie queue

Cette librairie contient toutes les fonctions et procédures qui permettent de traiter les fichiers contenus dans les files d'attente (cf. [Figure 16 : Élément d'une file d'attente](#)), ainsi que le fichier de configuration des queues (cf. [Figure 19 : Le fichier de configuration des queues](#)).

```

/* Record describing one job in a queue */
typedef struct job
{
    char username[MAX_USERNAME] ;
    char nmt[MAX_NT] ;
    int node ;
    bool life ;
    int Tmin ;
    bool reboot ;
    bool mail ;
} job ;

```

Les définitions en langage C des principales fonctions et procédures de cette librairie sont :

```

/* Return Tmax in minutes for the queue nq */
int tmax(int nq);

/* Change the value of Tmax for nq */
void config_q(int Tmax, int nq);

/* Read in filename data and return j */
void fileinjob(char *filename, job *j);

/* Insert job in queue */
void insert(job, int);

/* Return the number of files of rep */
int nb_files(char *rep);

/* Put in t the list of file's names of rep */
void dirintab(char *rep, char t[][MAXNF]);

/* Compare line i of t with line j by creation_date (cd)
 * Return 1 if cd of t[i] > cd of t[j] */
int compare_file(char t[][MAXNF], int i, int j);

/* Exchange line i with line j in t */
void exchange_file(char t[][MAXNF], int i, int j);

/* Sort by creation_date the files in t */
void sort_dir(char t[][MAXNF], int , int);

/* Return all jobs sorted by priority in t in function of ts_username
 * Return the number of jobs in queues */
int sort_queues(char t[][MAXNF], char *ts_username);

/* Return 1 and ac if there is a job in execution
 * Return 0 else */
int jobac(job *ac);

```

La librairie cgic

Cette librairie a été prise sur Internet à l'adresse suivante : <http://www.boutell.com/cgic/>
 Cette librairie sert d'interface entre un programme C et le monde WWW. En particulier, elle facilite la récupération de la saisie de champs sur un navigateur par une méthode GET ou POST.

III.6.2 L'algorithme de tri

Il est très souvent nécessaire de faire du tri dans le JMS, que ce soit pour le tri des files d'attente ou le tri du fichier calendrier.

L'algorithme de tri utilisé est l'algorithme du tri rapide (Quick Sort) rappelé ci-dessous :

```

procédure triRapide (entrée-sortie t : Séquence)
    triRapideRécuratif(t,1,longueur(t))
fproc triRapide

procédure triRapideRécuratif(entrée-sortie t : Séquence, entrée p,r :
Naturel)
    i, d : Naturel

    si p<r alors
        échanger(t, p, (p+r)/2)
        d = p
        pour i de p+1 à r
            si t[i] < t[p] alors
                d = d+1
                échanger(t, d, i)
            fsi

        fpour
            échanger(t, p, d)

            triRapideRécuratif(t, p, d-1)
            triRapideRécuratif(t, d+1, r)

        fsi
fproc triRapideRécuratif
    
```

C'est un algorithme récursif. Deux fonctions de comparaison ont donc été écrites (comparaison de deux lignes du fichier calendrier et comparaison de deux fichiers par leur date de création).

III.6.3 Les locks

De nombreux fichiers ou répertoires peuvent être consultés et modifiés au même moment si un grand nombre d'utilisateurs utilisent le JMS en même temps. Supposons par exemple, qu'à un instant donné, plusieurs utilisateurs insèrent des applications dans les files d'attente. Supposons de plus qu'à ce moment là, l'exécutif soit en train de parcourir les files d'attente pour déterminer l'application la plus prioritaire. Cela peut poser de gros problèmes dans la mesure où un répertoire associé à une file d'attente peut être lu et modifié au même instant.

Un autre problème du même type peut se poser : supposons que les nœuds de calcul aient été redémarrés et que pendant ce temps, l'exécutif décide de lancer l'exécution d'une application sur un des nœuds. Ce problème est similaire au précédent : il met en jeu la « concurrence » entre les différents exécutables du JMS.

Afin d'éviter ces conflits, la solution qui a été adoptée est la suivante : le JMS utilise un fichier de lock à chaque fois que cela est nécessaire. Qu'est-ce que cela signifie ? Dès qu'un exécutable (C ou CGI) est susceptible, par ses actions, de créer des conflits avec un autre exécutable, il utilise un fichier de lock. Ce fichier est appelé *lock* et se trouve dans le répertoire *conf*. Quand un exécutable commence son exécution, il ouvre ce fichier. Deux cas peuvent alors se produire :

- Soit **le fichier est libre**, c'est à dire qu'il n'y a pas d'autre exécutable en cours d'exécution : l'exécution peut alors se poursuivre et le fichier *lock* est fermé.
- Soit **le fichier est fermé**, c'est à dire qu'un autre exécutable est en cours d'exécution et que des conflits pourraient se produire : l'exécution est alors stoppée jusqu'à ce que le fichier de lock soit libéré. Le fichier de lock n'est libéré par un exécutable que lorsque son exécution se termine.

Ce système permet d'éviter les conflits mais il y a un inconvénient majeur. Si un exécutable utilisant le lock ne se termine pas pour une raison ou pour une autre, tout le JMS est bloqué.

III.6.4 La sécurité

III.6.4.1 Le bit SUID

Un grand nombre d'exécutables du JMS nécessite d'avoir les droits du superutilisateur (**root**). Or, les utilisateurs n'ont pas ces droits. En effet, par exemple, pour écrire dans les files d'attente, il faut avoir les droits du superutilisateur sinon n'importe quel utilisateur pourrait supprimer des queues les applications des autres utilisateurs et cela n'est évidemment pas permis. La solution retenue pour contourner ce problème est d'utiliser **le bit SUID (s)** pour tous les exécutables qui le nécessite.

Ce bit permet de forcer l'exécution avec l'identité (**UID**) du propriétaire du fichier et non pas avec celle de l'utilisateur ayant demandé l'exécution de ce fichier. Cela permet d'attribuer des droits supplémentaires de façon ponctuelle et contrôlée. Ce procédé ne met donc pas en cause la sécurité du JMS.

En revanche, le bit SUID n'est pas utilisé pour les shells UNIX pour des raisons de sécurité.

III.6.4.2 La sécurité et l'interface CGI

La connexion via Internet à une machine doit être sécurisée. Il faut, par exemple, empêcher que n'importe qui puisse lancer des applications. Pour cela, lorsqu'un utilisateur veut utiliser le JMS en utilisant l'interface CGI et donc, par ce biais, se connecter à la machine MPC, il doit être authentifié par un **login** et un **mot de passe**.

Cela est facilement réalisable, et de manière sûre, en utilisant le démon HTTP de Apache. Il faut utiliser un fichier **.htaccess** et un fichier **htpasswd.users** qui sont placés dans le répertoire *htdocs* du JMS (cf. *Annexe 3 : Manuel d'installation*). Seuls les utilisateurs ayant un login et un mot de passe pour utiliser le JMS de la machine MPC pourront alors se connecter via Internet. L'administrateur du JMS doit gérer ces utilisateurs.

Par ailleurs, les exécutable CGI sont exécutés par le démon HTTP. Afin que l'exécution des CGI ne soit pas détournée, un compte UNIX doit être créé pour disposer d'un utilisateur spécial qui ne serve qu'à exécuter les CGI. Le champ **LOGIN_HTTPD** du fichier de configuration *config* (cf. *Figure 18*) sert à préciser le login de cet utilisateur particulier. Il faut également configurer le serveur *Apache* pour que les CGI soient exécutés par cet utilisateur.

Dès lors, lorsque des utilisateurs exécuteront des CGI via Internet, ces exécutable seront, en fait, exécutés par l'utilisateur **LOGIN_HTTPD**. Au début de chacun des exécutable CGI, on teste si l'**UID** (user identity) est effectivement **LOGIN_HTTPD**. Si cela n'est pas le cas, le CGI s'arrête immédiatement.

Cependant, il est parfois nécessaire que certains CGI soient exécutés en tant que l'utilisateur qui s'est connecté à la machine. Pour cela, on utilise encore un bit SUID qui permet de passer de l'UID correspondant à l'utilisateur **LOGIN_HTTPD** à l'UID de l'utilisateur du JMS. Ce changement d'UID se fait au moyen de l'appel système **setreuid()**.

Enfin, il est à noter qu'aucun shell UNIX n'est exécuté directement via Internet. Un wrapper en C est systématiquement utilisé pour exécuter ces shells.

Chapitre III : Conclusions

I.	DÉROULEMENT DU STAGE	52
II.	APPORTS DU STAGE	52
III.	LES RÉUNIONS	53
IV.	LES ÉVOLUTIONS DU JMS-MPC	53
V.	LE PROJET DE RECHERCHE MPC	53

Chapitre 3 : Conclusions

Il est maintenant temps de faire un bilan global du stage que j'ai effectué au département ASIM du Laboratoire d'Informatique de Paris 6.

I. Déroulement du stage

Le stage s'est déroulé de la manière suivante. Tout d'abord, il m'a fallu prendre contact avec le milieu universitaire. Cela m'a permis, en particulier, de découvrir ce qu'est réellement un laboratoire de recherche de l'ampleur du LIP6, qui est l'un des plus grand laboratoire de recherche en informatique de France.

La deuxième étape a été de bien cerner le sujet de mon stage, mais aussi en grande partie, de le définir. Pour cela, j'ai passé un peu plus d'un mois à faire des recherches sur les Job Management System existants comme cela a été présenté à la section *Chapitre 2 : I. Généralités sur les JMS*. Ces recherches se sont faites, pour l'essentiel, sur Internet.

Ensuite, il a fallu fournir des spécifications du JMS pour la machine MPC (cf. *Chapitre 2 : II. Spécifications du JMS-MPC*). Avant d'arriver aux spécifications finales, il a eu plusieurs étapes afin de se rapprocher le plus possible, des réels besoins des utilisateurs. Cette étape de spécifications a duré environ 1 mois.

La quatrième étape correspond à la partie développement du JMS pour la machine MPC, ce qui a représenté un peu moins de deux mois de travail.

Enfin, il y a eu une période de tests et surtout de mises au point qui a duré environ un mois.

II. Apports du stage

Ce stage m'a été bénéfique à de nombreux points de vue. Outre la prise de connaissance du monde de la recherche que j'ai toujours souhaité côtoyer, il m'a permis de conduire un projet du début à sa fin. J'ai en particulier pu voir toutes les étapes qui sont nécessaires à la conception d'un outil logiciel comme l'est le JMS pour la machine MPC.

Par ailleurs, d'un point de vue technique, ce stage m'a permis d'élargir mes connaissances du langage C mais aussi du monde UNIX. En particulier, j'ai pris conscience des problèmes liés au changement d'UID. J'ai réalisé que les problèmes de sécurité sont très importants lorsqu'on réalise une application comme le JMS. Enfin, j'ai appris beaucoup de choses sur les problèmes liés à l'interface entre Internet et une machine comme celle du LIP6, en particulier par le biais de la programmation CGI.

D'autre part, j'ai également appris à communiquer avec des personnes à l'étranger pour obtenir des informations importantes.

III. Les réunions

J'ai assisté à plusieurs types de réunion dans le cadre du projet MPC. Il s'agit, tout d'abord, de réunions internes au département ASIM qui regroupent tous les chercheurs de l'équipe dont les travaux de recherche sont en liaison avec la machine MPC. Ces réunions sont au nombre de deux par mois environ.

D'autre part, j'ai participé aux réunions mensuelles regroupant tous les chercheurs de France qui travaillent sur le projet MPC.

IV. Les évolutions du JMS-MPC

Le JMS pour la machine MPC peut évoluer selon deux directions. La première serait l'importation du JMS sur une plate-forme autre qu'UNIX FreeBSD. En effet, pour l'instant, les machines MPC fonctionnent sous le système d'exploitation FreeBSD mais des projets sont en cours pour porter une partie des couches logicielles sous LINUX. Le portage du JMS sur d'autres systèmes UNIX ne devrait faire intervenir que des changements mineurs.

La deuxième évolution serait de faire en sorte que le JMS puisse lancer non seulement, des applications PVM-MPC mais aussi d'autres types d'application comme par exemple des applications MPI-MPC. Mais cela suppose que le portage de *MPI* (MPI est un environnement de programmation parallèle) soit réalisé sur la machine MPC. Dans ce domaine, des projets sont également en cours. Les modifications à faire pour intégrer cette évolution au sein du JMS ne devraient pas être compliquées.

Plus généralement, il faudra que le JMS suivent les évolutions du projet MPC.

V. Le projet de recherche MPC

Le département ASIM du LIP6 est fortement impliqué, depuis 1993, dans le développement d'une technologie d'interconnexion haute performance qui est devenue le standard IEEE 1355. Le département ASIM a été sollicité par différents industriels (Bull, SGS-Thomson, Parsytec, Thomson) pour participer à quatre projets européens visant le développement ou l'exploitation de cette technologie HSL. Les composants VLSI RCUBE et PCI-DDC, qui ont été entièrement conçus à l'Université Pierre et Marie Curie, sont actuellement utilisés dans plusieurs machines industrielles.

Le projet MPC, qui a démarré en janvier 1995 sous la responsabilité de Alain Greiner, possède un fort impact puisqu'il existe des plates-formes MPC à Versailles (PRISM), Evry (INT), Toulouse, Amiens, etc.

Les cartes FastHSL et les composants VLSI RCUBE et PCI-DDC sont commercialisés par la société Tachys Technologies, qui est une « start-up » du département ASIM.

Le projet MPC a donc encore beaucoup d'avenir devant lui. Je souhaite continuer à travailler sur ce projet, dans le cadre de mon stage de DEA d'une part, et dans le cadre éventuel d'une thèse d'autre part.

Bibliographie

I. Sites Internet

Site de l'Université de Paris 6	http://www.admp6.jussieu.fr/
Site du LIP6	http://www.lip6.fr/
Site du département ASIM	http://www-asim.lip6.fr/
Site de la machine MPC du LIP6	http://mpc.lip6.fr/ ou http://www-asim.lip6.fr/mpc
Site du JMS CODINE	http://www.genias.de/welcome.html
Site du JMS NQE	http://www.sgi.com/software/nqe/
Site du JMS GNU Queue	http://bioinfo.mbb.yale.edu/fom/cache/1.html
Site du JMS LSF	http://www.lerc.nasa.gov/WWW/LSF/lfs_homepage.html
Site du JMS Condor	http://www.cs.wisc.edu/condor/
Site du JMS DQS	http://www.msi.umn.edu/bscl/info/dqs/
Site de la librairie CGIC	http://www.boutell.com/cgic/
Site du serveur HTTP Apache	http://www.apache.org/

II. Documentations et ouvrages

Emulateur logiciel MPC/1 Manuel d'installation et d'utilisation version 3.0
Alexandre Fenyö, Frédéric Potter, Pierre David, Décembre 1996

Portage de PVM sur MPC Rapport périodique d'activité
Karim Mana, Octobre 1998

Scalable parallel computing (WCB/Mc Graw Hill)
Kai Hwang et Zhiwei Xu, 1998

NQE User's Guide

Cgic : an ANSI C library for CGI Programming
Thomas Boutell

Le langage C (Masson)
B.W. Kernighan & D.M. Ritchie, 2^{ème} édition

UNIX, Guide de l'utilisateur (Ellipses)
D. Bouillet

La programmation sous UNIX (Ediscience)
J.M. Rifflet, 3^{ème} édition

Annexes :

ANNEXE 1 : L'ENVIRONNEMENT DE PROGRAMMATION PARALLÈLE PVM **57**

I.	INTRODUCTION	57
II.	COMMUNICATIONS	57
III.	FONCTIONNEMENT GÉNÉRAL	57
IV.	VISION CONCEPTUELLE DU SYSTÈME PVM	58

ANNEXE 2 : PVM POUR MPC (COMPLÉMENTS) **59**

I.	CRÉATION D'UNE TÂCHE PVM SUR LA MACHINE MPC	59
II.	COMMUNICATION ENTRE UNE TÂCHE DE CALCUL ET UNE TÂCHE MAÎTRE	60
III.	COMMUNICATION ENTRE DEUX TÂCHES DE CALCUL DISTANTES	60
IV.	UN EXEMPLE D'APPLICATION PVM POUR MPC	62
	IV.1 CODE SOURCE D'UNE APPLICATION PVM-MPC	62
	IV.1.1 Code de master1.c	62
	IV.1.2 Code de slave1.c	63
	IV.2 COMPILATION D'UNE APPLICATION PVM-MPC	65

ANNEXE 3 : MANUEL D'INSTALLATION DU JMS **66**

ANNEXE 4 : GUIDE D'UTILISATION DU JMS **67**

I.	PRÉALABLES	67
II.	L'INTERFACE LIGNE DE COMMANDE	68
	II.1 LES COMMANDES UTILISATEUR	68
	II.2 LES COMMANDES ADMINISTRATEUR	69
III.	L'INTERFACE CGI	69
	III.1 LES COMMANDES UTILISATEUR	69
	III.2 LES COMMANDES ADMINISTRATEUR	78

Annexe 1 : L'environnement de programmation parallèle PVM

I. Introduction

PVM est constitué d'une librairie et d'un démon. C'est un système de communication permettant aux applications d'utiliser un ensemble de machines, potentiellement hétérogènes et de puissances variables, interconnectés par un réseau de communication, rendant ainsi les communications indépendantes du système d'exploitation.

PVM a été développé à la fin des années 80 à l'Oak Ridge National Laboratory pour le Département de l'énergie Américaine et atteint aujourd'hui une large popularité. Il est considéré comme un standard pour l'écriture de programmes parallèles.

II. Communications

Les applications lancées sur PVM répartissent des tâches sur les différents ordinateurs constituant la machine parallèle. PVM garantit que chaque tâche de la machine virtuelle peut émettre un nombre quelconque de messages vers tout autre tâche de la machine. L'échange de message entre tâches PVM est réalisé grâce à deux primitives de base : *pvm_send()* et *pvm_recv()*. La première est exécutée par l'émetteur et a pour effet de transmettre les données vers le récepteur et ceci en précisant dans les arguments uniquement le numéro de tâche du récepteur ainsi qu'un « flag » d'identification que le récepteur doit bien évidemment connaître. Ce dernier ne prendra connaissance des données que s'il exécute la deuxième primitive. Entre ces deux commandes, le système se charge de préparer les données, de les transférer vers le site destinataire puis de les remettre à leur destinataire final. Le transfert est indépendant des deux tâches qui s'exécutent.

PVM offre un ensemble de variantes des deux primitives précédentes permettant chacune de réaliser un type de communication mais dans le cas présent, c'est-à-dire d'une implémentation à démon unique, les appels sont tous bloquants sur les machines autre que celle du démon.

III. Fonctionnement général

L'ensemble des tâches est distribué sur des machines hôtes qui constituent la machine virtuelle. Ces tâches sont gérées par un ou plusieurs démons selon le type d'implémentation. Dans le cas de la machine MPC, la configuration est à démon unique, comme pour le cas de la machine parallèle Paragon d'Intel, qui gère l'ensemble des tâches de la machine virtuelle comme si elles étaient locales à son site. Ce type d'implémentation se retrouve dans les machines massivement parallèles qui disposent d'un réseau d'interconnexion rapide.

Dans cette configuration, nous distinguons la machine portant le démon (nœud de service) des autres machines du réseau (nœuds de calcul). Cette distinction nous permet d'identifier les étapes effectuées pour un échange de message dans chacun des deux cas.

Lorsqu'une tâche, se trouvant sur la machine du démon, veut émettre un message, elle ne fait que l'insérer dans la file des messages à émettre, le confiant ainsi au démon. Ce dernier se charge alors de le router ou de le délivrer à sa tâche destinataire. La tâche émettrice est donc libre de retrouver son travail de fond.

Si une tâche destinataire est sur un nœud avec démon, celui-ci reçoit le message et l'insère dans la file des messages reçus pour la tâche en question. Cette dernière pourra alors en disposer lorsqu'elle effectuera un *pvm_recv()*.

En l'absence d'un démon PVM sur le site hôte de la tâche émettrice, celle-ci fait appel à la librairie PVM pour effectuer le transfert. De même, si une tâche réceptrice se trouvant sur un site sans démon exécute un *pvm_recv()* elle scrutera le buffer de dépôt du réseau jusqu'à la réception du message.

IV. Vision conceptuelle du système PVM

La bibliothèque PVM est constituée de deux principales parties (voir figure ci-dessous). La première est indépendante du système d'exploitation. Elle regroupe toutes les fonctions de gestion des tâches PVM, des groupes ainsi que les fonctions de communication de haut-niveau. La deuxième, de plus bas niveau, comprend les fonctions de transfert des paquets sur le réseau natif. Cette partie reconnaît le système de la machine hôte et s'adapte à son architecture. C'est à ce niveau-là que vont intervenir les communications avec le réseau HSL.

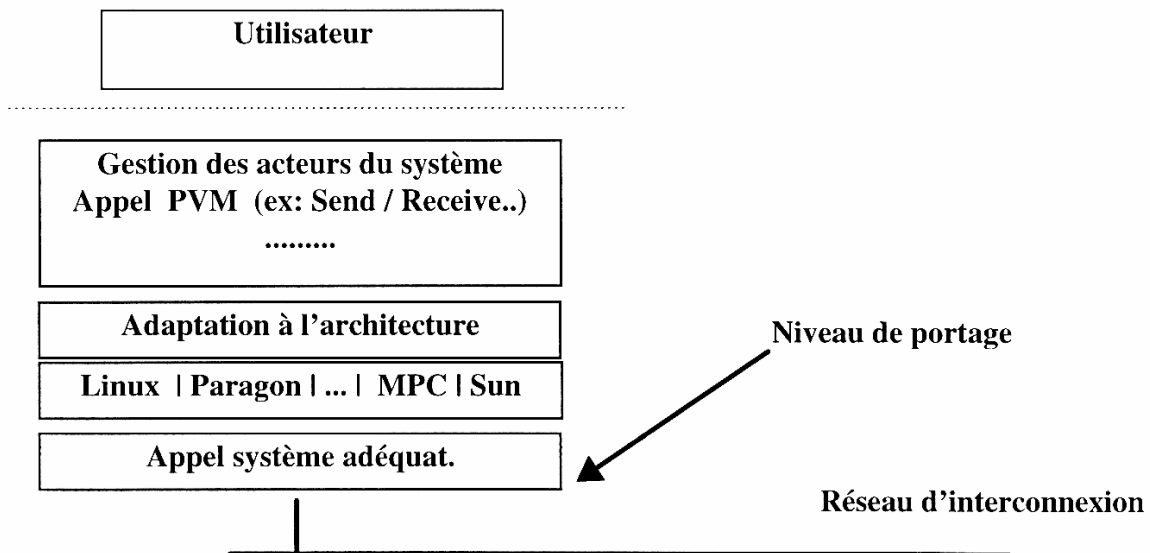


Figure 21 : Couches du système PVM

C'est pourquoi, il était nécessaire de faire un portage du PVM standard sur la machine MPC. Pour plus d'informations sur PVM : <http://www.epm.ornl.gov/pvm>

Annexe 2 : PVM pour MPC (compléments)

I. Création d'une tâche PVM sur la machine MPC

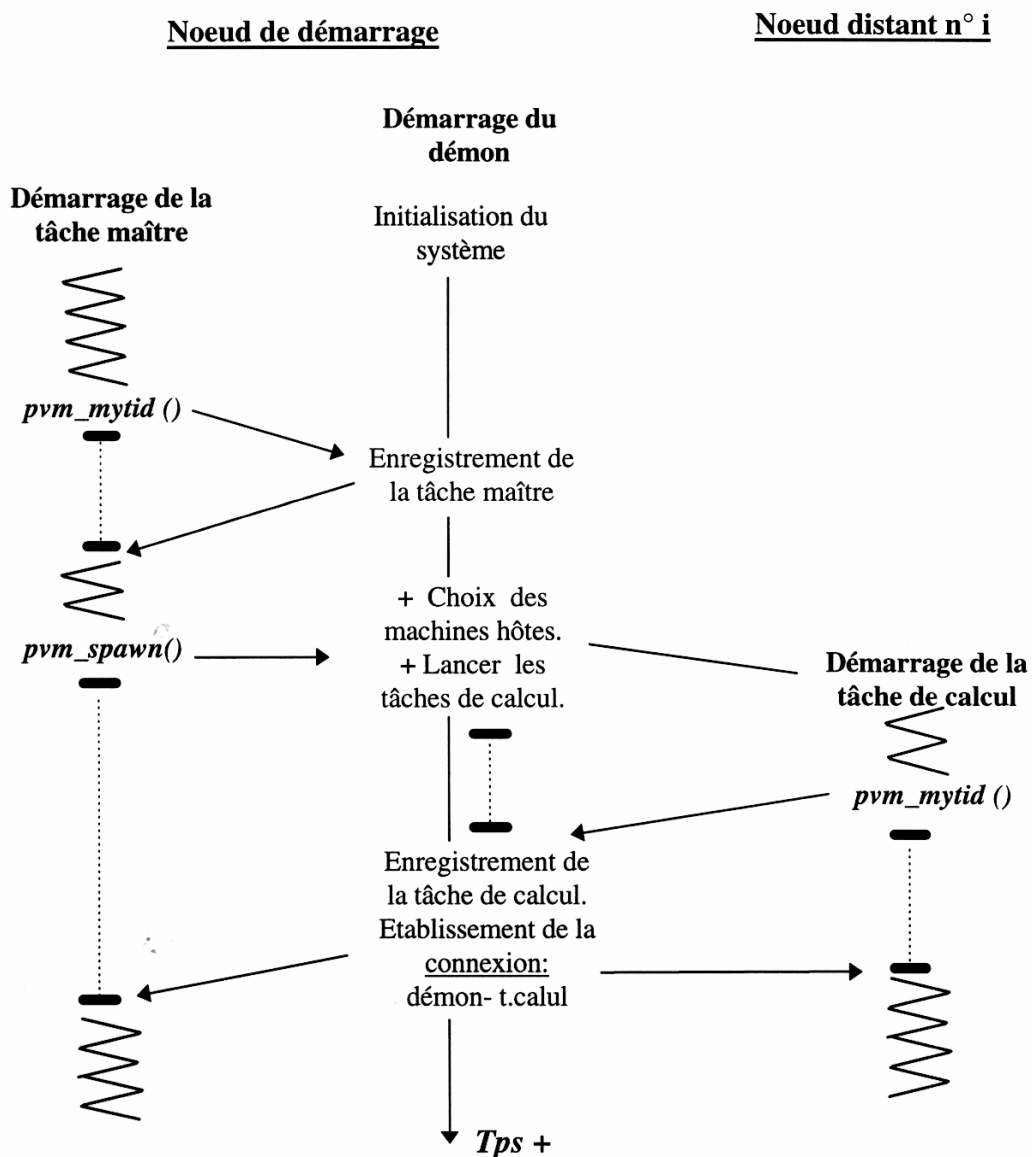


Figure 22 : Création et Initialisation d'une tâche PVM sur la machine MPC

II. Communication entre une tâche de calcul et une tâche maître

La tâche maître se trouve sur le nœud de service. Toutes ses communications avec les tâches distantes se feront via le démon. Les messages que le tâche maître veut émettre sont confiés à ce dernier qui les transfère à la tâche destinataire grâce à sa connexion HSL avec celle-ci. De même les messages destinés à la tâche maître sont reçus par le démon puis délivrés à cette dernière. **Un des inconvénients principaux de PVM-MPC est que les communications entre le démon et la tâche maître sont réalisées par Socket Unix locales ce qui ralentit considérablement les performances de la machine virtuelle** (cf. *Figure 23*).

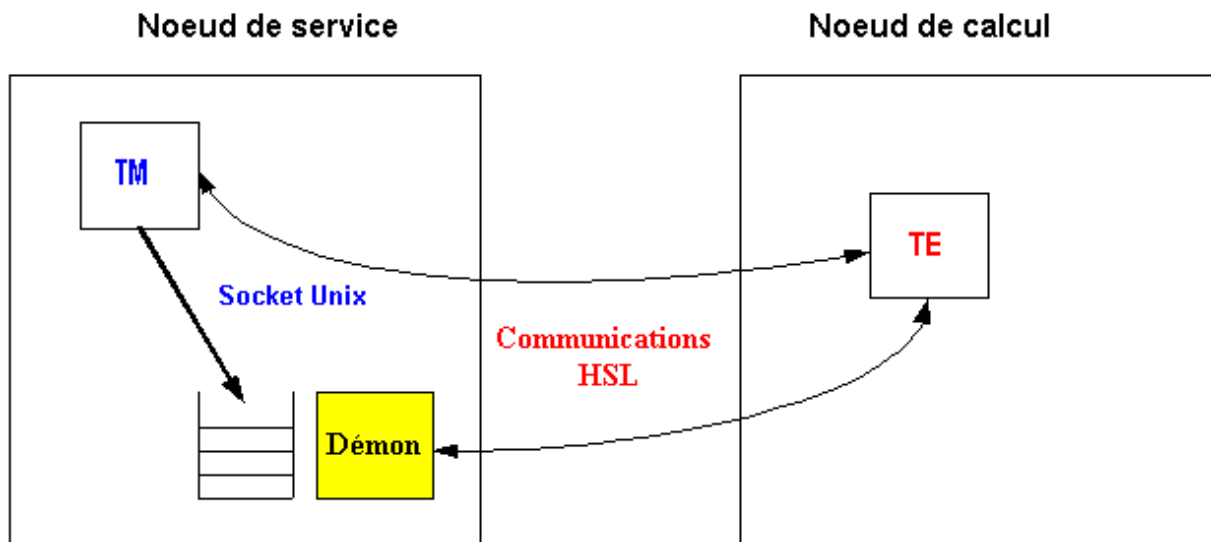


Figure 23 : Un exemple de communication maître-esclave

Dans l'exemple ci-dessus, la tâche maître exécute un `pvm_send()` et la tâche esclave un `pvm_recv()`. La communication HSL permet d'écrire directement dans les buffers de la tâche esclave.

Le passage systématique par les sockets Unix pour les communications avec la tâche maître pourrait constituer un goulot d'étranglement pour le reste du système.

III. Communication entre deux tâches de calcul distantes

La première opération consiste à établir une connexion directe entre les deux tâches. Cette connexion se fera via le démon. Après avoir créé deux tampons verrouillés en mémoire physique sur la tâche émettrice (un pour l'émission et un pour la réception), cette dernière envoie une requête de connexion au démon portant le TID de la tâche à laquelle elle veut se connecter. Elle se met alors en attente d'une réponse de la part du démon. Ensuite, elle attend une acceptation de la connexion de la part de la tâche distante concernée. De son côté le démon reçoit la première requête, attribue le canal de connexion, renvoie à la première tâche

le numéro du canal de connexion et envoie un message à la seconde avec le numéro de ce canal et le TID de la tâche désirant se connecter. Le démon alloue 2 tampons nécessaires à la connexion à la tâche réceptrice puis indique à la tâche émettrice qu'elle peut émettre. Chacune des deux tâches garde trace de cette connexion et l'utilise pour ses communications tout au long de sa vie.

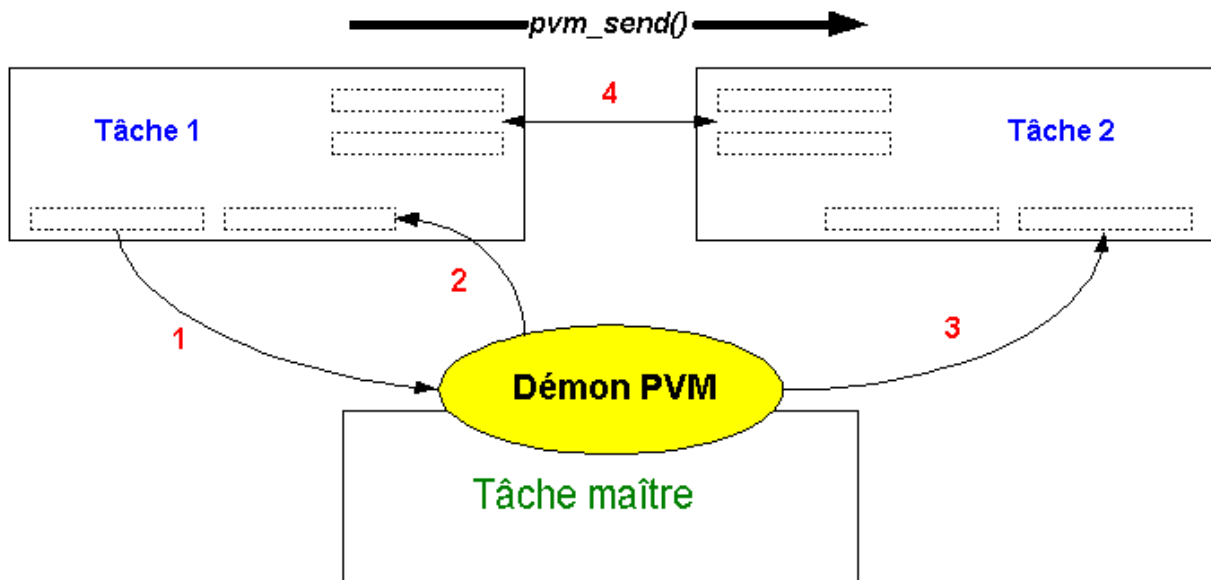


Figure 24 : Protocole de connexion HSL entre 2 tâches esclaves

Les messages PVM sont fragmentés pour être transférés sur le réseau HSL. La primitive **slrpv_send** de la couche logicielle SLRPV est utilisée pour transmettre les fragments PVM. Ces données sont recopiées directement sur le nœud destinataire et sont délivrées à la tâche distante grâce à la primitive **slrpv_rcv**.

Lorsque les deux tâches de calcul sont sur un même nœud, la mise sur le réseau des données est remplacée par une recopie des données au niveau local. Deux buffers sont alors nécessaires (au lieu de 4 pour des communications distantes) ; il y a un buffer par sens de communication. Cependant, la connexion entre deux tâches locales s'établit toujours par l'intermédiaire du démon.

IV. Un exemple d'application PVM pour MPC

IV.1 Code source d'une application PVM-MPC

Voici un exemple très simple d'application PVM pour MPC. Il se décompose en 2 exécutables : le maître *master1* qui doit être lancé sur le nœud de service et les esclaves *slave1* qui sont lancés sur les nœuds de calcul par le démon grâce à la fonction *pvm_spawn()*.

IV.1.1 Code de master1.c

```

/* Creation de taches de calculs recuperations des resultats */

#include <stdio.h>
#include "pvm3.h"
#define SLAVENAME "slave1"

main()
{
    int mytid;          /* my task id */
    int tids[24];      /* slave task ids */
    int n, nproc, numt, i, who, msgtype;
    float data[100], result[24];

    /* enroll in pvm */
    mytid = pvm_mytid();
    printf("My tid : %d\n",mytid);

    /* Set number of slaves to start */
    /* Can not do stdin from spawned task */
    if ( pvm_parent() == PvmNoParent )
    {
        puts("How many slave programs (1-24)?");
        scanf("%d", &nproc);
    }

    /* start up slave tasks */
    numt=pvm_spawn(SLAVENAME, (char**)0, 0, "", nproc, tids);
    if( numt < nproc )
    {
        printf("Trouble spawning slaves. Aborting. Error codes are:\n");
        for( i=numt ; i<nproc ; i++ )
        {
            printf("TID %d %d\n",i,tids[i]);
        }
        for( i=0 ; i<numt ; i++ )
        {
            pvm_kill( tids[i] );
        }
        pvm_exit();
        exit();
    }
}

```

```

/* Begin User Program */
n = 100;
for ( i=0 ; i<n ; i++ ) data[i] = 1;

/* Broadcast initial data to slave tasks */

pvm_initsend(PvmDataDefault);
pvm_pkint(&nproc, 1, 1);
pvm_pkint(tids, nproc, 1);
pvm_pkint(&n, 1, 1);
pvm_pkfloat(data, n, 1);

for (i=0; i<nproc ; i++) pvm_send(tids[i],0);

/* Wait for results from slaves */
msgtype = 5;
for( i=0 ; i<nproc ; i++ )
{
    pvm_rcv( -1, msgtype );
    pvm_upkint( &who, 1, 1 );
    pvm_upkfloat( &result[who], 1, 1 );
    printf("I got %f from %d\n",result[who],who);
}
/* OK */
/* Program Finished exit PVM before stopping */
pvm_exit();
}

```

IV.1.2 Code de slave1.c

```

#include <stdio.h>
#include "pvm3.h"

FILE *fdlog;
char txt[128];
main()
{
    int res;
    int mytid; /* my task id */
    int tids[32]; /* task ids */
    int n, me, i, nproc, master, msgtype;
    float data[100];
    char logfile[30];

    sprintf(logfile, "/tmp/pvmlib.log.sla01.%d", (int) getpid());
    fdlog = fopen(logfile, "w+");
    if (!fdlog) pvmlogperror("ERROR : Log file");

    setvbuf(fdlog, (char *)NULL, _IONBF, 0);
    pvmlogerror("Log file OK!\n");
    pvmlogerror("slave OK\n");

    /* enroll in pvm */
    mytid = pvm_mytid();

    /* Receive data from master */
    msgtype = 0;

```

```
pvmlogerror("Before pvm_recv\n");
res=pvm_recv( -1, msgtype );
pvm_upkint(&nproc, 1, 1);
sprintf(txt,"Receive : number processors : %x \n",nproc);
pvmlogerror(txt);

pvm_upkint(tids, nproc, 1);
sprintf(txt,"Receive : tid : %x \n",tids[0]);
pvmlogerror(txt);

pvm_upkint(&n, 1, 1);
sprintf(txt,"Receive : n : %d \n",n);
pvmlogerror(txt);

pvm_upkfloat(data, n, 1);
sprintf(txt,"Receive : data : %f \n",data[0]);
pvmlogerror(txt);

/* Determine which slave I am (0 -- nproc-1) */
for( i=0; i<nproc ; i++ )
    if( mytid == tids[i] )
    {
        me = i;
        sprintf(txt,"me : %d \n",me);
        pvmlogerror(txt);
        break;
    }

/* Send result to master */
pvm_initsend( PvmDataDefault );
pvm_pkint( &me, 1, 1 );
pvm_pkfloat( &data, 1, 1 );
msgtype = 5;
master = pvm_parent();
sprintf(txt,"Slave : send - master : %x \n",master);
pvmlogerror(txt);
pvmlogerror("pvm send\n");
pvm_send( master, msgtype );

/* OK */
pvmlogerror("Pvm exit\n");
/* Program finished. Exit PVM before stopping */
pvm_exit();

fprintf(fdlog,"End of program... slave1\n");
fclose(fdlog);
exit(0);
}
```


IV.2 Compilation d'une application PVM-MPC

Voici, par exemple, le type de Makefile qu'il faut utiliser. Dans le Makefile suivant, la tâche maître est *master1* et la tâche esclave est *slave1*.

```

PVM_ARCH = MPC
ARCHLIB = -lrpcsvc

PVMDIR = ../..
PVMLIB = $(PVM_ROOT)/lib/$(PVM_ARCH)/libpvm3.a
PVMPELIB = $(PVMDIR)/lib/$(PVM_ARCH)/libpvm3pe.a
SDIR = ..
BDIR = $(PVM_ROOT)/bin
XDIR = $(BDIR)/$(PVM_ARCH)

CC = cc
CFLAGS = -g -I../..//include
LIBS = $(PVMLIB) $(ARCHLIB)
NODELIBS = $(PVMPELIB) $(ARCHLIB)

default: master1 slave1

$(XDIR): $(BDIR)
- mkdir $(XDIR)

$(BDIR):
- mkdir $(BDIR)

clean:
rm -f ../..//bin/MPC/* *.o

#----->>>>>

master1: $(SDIR)/master1.c $(XDIR)
$(CC) $(CFLAGS) -o master1 $(SDIR)/master1.c
$(LIBS)
mv master1 $(XDIR)

slave1: $(SDIR)/slave1.c $(XDIR)
$(CC) $(CFLAGS) -o slave1 $(SDIR)/slave1.c
$(NODELIBS)
mv slave1 $(XDIR)

```

Annexe 3 : Manuel d'installation du JMS

Voici très brièvement les étapes à suivre pour installer le JMS de la machine MPC : une autre documentation plus complète sera bientôt disponible.

- Système d'exploitation : **UNIX FreeBSD 3.1**
- L'ensemble des packages doit contenir : **zsh, rsh, ssh** et **apache**
- Installer sur tous les nœuds de calcul et la console, la distribution de **MPC-OS** (par Alexandre FENYO)
- Installer sur tous les nœuds et la console, la distribution de **PVM pour la machine MPC**
- Récupérer et décompresser sur tous les nœuds et la console, la distribution **MPC-JMS**
- Positionner une variable d'environnement **\$JMSDIR** correspondant au répertoire d'installation du JMS.
- Configurer les fichiers **config, config_queues** et **calendar** se trouvant dans le répertoire **\$JMSDIR/conf**
- Compiler la distribution du JMS : taper **gmake** dans le répertoire **\$JMSDIR/src**
- Mettre dans la variable d'environnement **\$PATH**, le répertoire **\$JMSDIR/bin** afin d'avoir accès aux exécutable de la ligne de commande du JMS
- Configurer le serveur apache pour que les exécutions des CGI soient faites par l'utilisateur **wwwcgi**
- Créer un fichier **.htaccess** dans le répertoire **\$JMSDIR/htdocs**
- Créer des comptes utilisateurs
- Ajouter les utilisateurs du JMS dans le fichier **\$JMSDIR/htdocs/htpasswd.users** grâce à la commande **htpasswd**

Annexe 4 : Guide d'utilisation du JMS

Le Job Management système pour la machine MPC permet de disposer de plusieurs fonctionnalités pour utiliser la machine MPC. L'utilisateur peut accéder à ces fonctionnalités soit par l'interface ligne de commande, soit par l'interface CGI.

I. Préalables

Pour qu'un utilisateur puisse utiliser le JMS, il doit remplir les conditions suivantes :

- Avoir un compte UNIX sur la console de la machine MPC et sur tous les nœuds de calcul
- Il doit avoir sur chacun des nœuds de calcul un répertoire `~/MPC-JMS`. Ce répertoire doit lui même contenir un sous répertoire **nodes-bin** contenant tous les binaires PVM-MPC de l'utilisateur et un sous répertoire **logs** contenant les fichiers de log de ses applications PVM-MPC. Le répertoire MPC-JMS doit être le même sur tous les nœuds de calcul

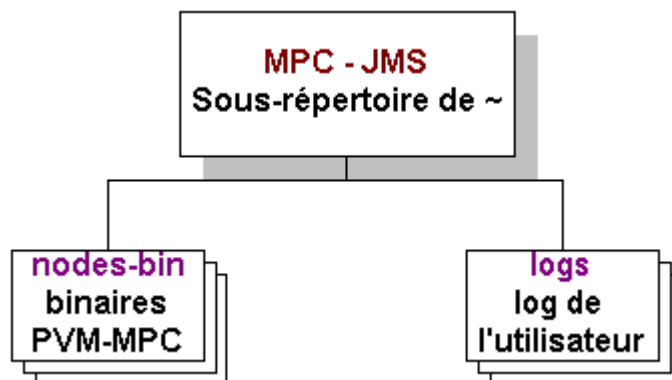


Figure 25 : Le répertoire MPC-JMS sur les nœuds de calcul

- L'utilisateur doit avoir une variable d'environnement **\$JMSDIR** positionnée sur le répertoire d'installation du JMS. Il doit, par ailleurs, contenir dans sa variable d'environnement **\$PATH**, le répertoire `$JMSDIR/bin` afin d'avoir accès aux exécutable de la ligne de commande du JMS
- Pour que l'utilisateur puisse utiliser l'interface CGI du JMS, un **login** et un **mot de passe** doivent lui avoir été attribués par l'administrateur du JMS

Enfin, il est conseillé aux utilisateurs non avertis de consulter la section [Chapitre 2 : II.3 Les fonctionnalités](#).

II. L'interface ligne de commande

Cette interface est utilisée aussi bien par les utilisateurs que par l'administrateur du JMS.

II.1 Les commandes utilisateur

L'utilisateur dispose, à partir de la ligne de commande, des fonctions suivantes :

Fonction	Commande associée
Insérer une application dans une queue	put_in_q nq (nq =1, 2 ou 3)
Enlever une application d'une queue	dqueue
Tuer l'application en cours d'exécution	killac
Afficher l'état des queues	stateq 'calend'
Afficher l'état du calendrier	statecalend
Afficher la période et l'utilisateur actuels	calend
Devenir l'utilisateur prioritaire	takets
Tester l'application en cours d'exécution	testac Tav (Tav = temps de test en minutes)
Liste des démons et drivers PVM-MPC	listdd.sh
Liste des processus sur tous les nœuds	psnode.sh
Rapatrier les fichiers de log	logs.sh

Figure 26 : Commandes utilisateur

- **nq** représente le numéro de la file d'attente dans laquelle l'utilisateur souhaite insérer son application. (nq = 1 pour la queue rapide, nq = 2 pour la queue moyenne et nq = 3 pour la queue lente)
- **stateq** affiche l'état des files d'attente en classant par ordre de priorité les applications en fonction de la période actuelle du calendrier
- **takets** permet de devenir l'utilisateur prioritaire à la condition que la période actuelle du calendrier soit de type USER et qu'aucun utilisateur n'ait encore réservé la période
- **Tav** représente le temps de test de l'application : si pendant Tav minutes, l'application n'engendre aucune communication HSL, alors l'utilisateur est averti par un mail.

II.2 Les commandes administrateur

L'administrateur dispose de toutes les commandes utilisateur (cf. paragraphe précédent). Afin d'exécuter les commandes précédentes en tant qu'un utilisateur du JMS, l'administrateur doit utiliser la commande UNIX **su**.

Par ailleurs, l'administrateur dispose en plus des commandes suivantes :

Fonction	Commande
Démarrer l'exécutif	start.sh
Arrêter l'exécutif	stop.sh
Effacer toutes les applications d'une queue	rmqueue.sh nq (nq =1 ou nq = 2 ou nq =3)
Configurer une queue	configq
Redémarrer tous les nœuds de calcul	reboot.sh

Figure 27 : Commandes administrateur


- **nq** représente le numéro de la file d'attente dans laquelle l'utilisateur souhaite insérer son application. (nq = 1 pour la queue rapide, nq = 2 pour la queue moyenne et nq = 3 pour la queue lente)
- **configq** permet de donner le paramètre **Tmax** pour une queue (temps maximal d'exécution d'une application issue de la queue en minutes)
- le reboot des nœuds de calcul ne fonctionne qu'à la condition qu'il y ait **REB = 1** dans le fichier de configuration **config**

III. L'interface CGI

III.1 Les commandes utilisateur

L'utilisateur accède à l'interface CGI à partir du WEB en rentrant l'URL de la machine MPC sous un navigateur quelconque. Il doit alors saisir immédiatement son **login** et son **mot de passe**. Il a alors accès à la page d'accueil du JMS (cf. *Figure 28 : Page d'accueil du JMS*)

Ensuite, il doit cliquer sur le bouton « **click here to enter** ». Il apparaît alors un menu qui lui permet d'accéder à toutes les fonctionnalités du JMS pour la machine MPC (cf. *Figure 29 : Menu utilisateur du JMS*). Ce menu se décompose en trois parties : les fonctions liées à l'exécution des applications, les fonctions concernant la consultation des objets du JMS et enfin le rapatriement des fichiers de log.



Asim
The team
LIP6 Laboratory
Paris, France



Job Management System for MPC

Welcome on MPC-JMS Web site !

Click here to enter

Web servers relative to MPC

- Internet Parallel Computing Archive
- IEEE 1355 association
- MPC general overview web server
- Workgroup on parallel computers (DEA SI)
- «Grappe» workgroup (IRISA)
- MPI PVM


Powered by



Powered by




Server design : Olivier Glück
Date: 1999/05/10 15:00:13.5




Copyright © 1999-2000 UPMC/LIP6
All rights reserved

Figure 28 : Page d'accueil du JMS



Asim team
LIP6 Laboratory
Paris, France



Job Management System for MPC


- Jobs**

Put a job in a queue	Click
Remove a job from a queue	Click
Kill the job in execution	Click
- Status**

State of queues	Click
State of calendar	Click
Become the priority user	Click
See the job in execution	Click
Test of the job in execution	Click
List of MPC-PVM daemons and drivers	Click
List of all your processes on all nodes	Click
- Logs**

Logs go back	Click
--------------	-------

Server design : Olivier Glück
Date: 1999/05/10 15:00:13.5



Copyright © 1999-2000 UPMC/LIP6
All rights reserved

Figure 29 : Menu utilisateur du JMS

L'utilisateur a le choix entre 11 fonctionnalités du JMS pour la machine MPC.

Insérer une application dans une queue

Quand l'utilisateur veut insérer une application dans une queue, il doit saisir les informations suivantes (cf. *Figure 30*) :

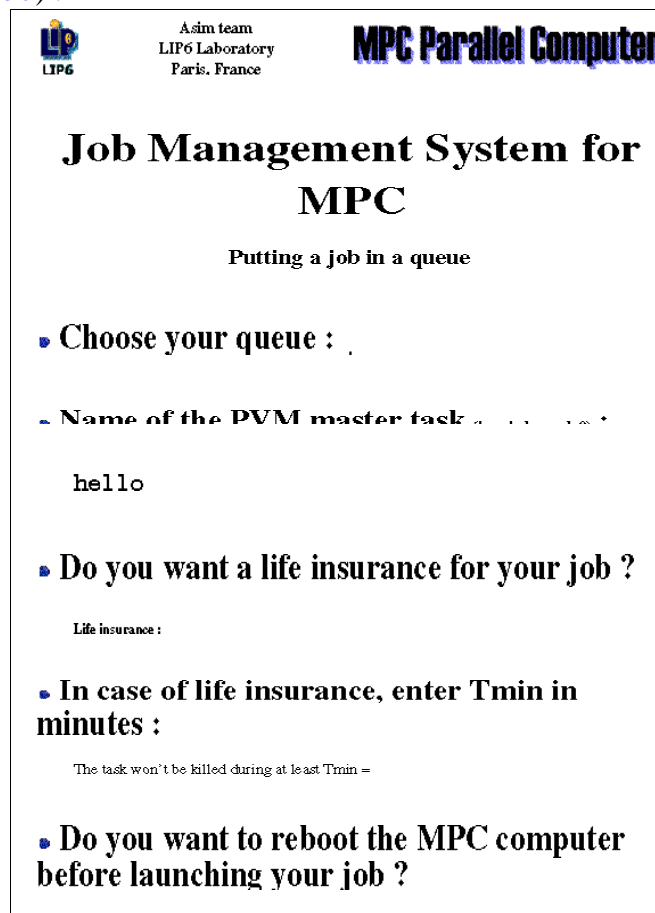


Figure 30 : Insérer une application dans une file

- Choix de la queue : **slow**, **medium** ou **quick**
- Nom de la tâche maître PVM-MPC, qui sera lancée sur le nœud 0 (nœud du démon)
- L'utilisateur doit cocher la case « **Life insurance** » s'il souhaite une assurance vie pour son application (par défaut, cette case n'est pas cochée)
- S'il souhaite une assurance vie, l'utilisateur doit spécifier le paramètre **Tav** (ou Tmin). Il représente le temps de test de l'application : si pendant Tav minutes, l'application n'engendre aucune communication HSL, alors l'utilisateur est averti par un mail.
- **Reboot** : l'utilisateur doit cocher cette case s'il souhaite redémarrer tous les nœuds de calcul avant l'exécution de son application. Le reboot ne sera effectif qu'à la condition que le JMS l'autorise (voir l'administrateur). Par défaut, cette case n'est pas cochée.
- **Mail report** : cocher cette case pour recevoir un mail lorsque, votre application est lancée ou bien votre application est tuée par le JMS. Par défaut, cette case est cochée.

Pour valider la saisie, cocher sur la case « **register** ». Il apparaît alors toutes les informations relatives à l'application que l'utilisateur vient d'insérer dans une queue (cf. *Figure 31*).

Job Management System for MPC

Putting a job in a queue

Report

Put in queue : OK

You have put the PVM master task *hello* under the login *olivier*.

Node : 0
 Queue : 3
 Life insurance : 1
 Request : 4
 Mail report : 1




Figure 31 : Résultat de l'insertion

Supprimer une application d'une file d'attente

Job Management System for MPC

Remove a job from a queue

List of your jobs in queues sorted by priority

Number	Queue	Date	Hour	PVM	Master Task
1	2	1999/06/23	12:07:55		ping_pong
2	3	1999/06/23	12:07:07		hello_world

• Which job do you want to remove (1-2) ?

• Be careful : you are going to remove a job from a queue on MPC computer

Remove job




Figure 32 : Supprimer une application d'une queue

Un utilisateur ne peut supprimer que des applications lui appartenant. Celles-ci sont numérotées à partir de 1. Il doit donner le numéro de l'application qu'il souhaite supprimer puis cliquer sur le bouton « **Remove job** ».

Tuer l'application qui est en cours d'exécution

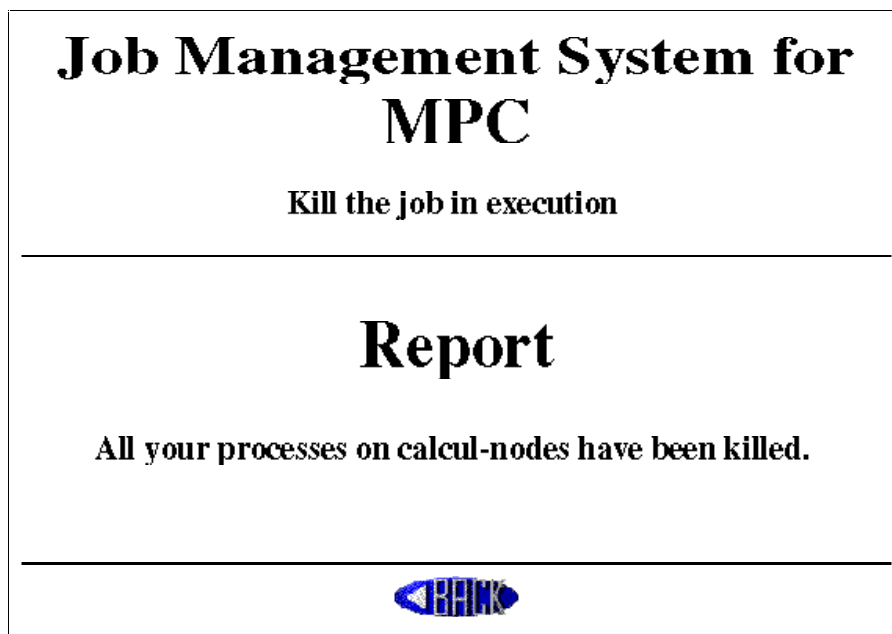


Figure 33 : Tuer son application

Un utilisateur ne peut tuer qu'une application lui appartenant. **Cela revient en fait, à tuer tous les processus de l'utilisateur sur tous les nœuds de calcul.**

Consulter l'état des files d'attente

Cela permet d'afficher le contenu de toutes les files d'attente selon l'ordre des priorités actuelles définies par le calendrier. Les applications appartenant à l'utilisateur sont affichées en gras. L'utilisateur peut ainsi se rendre compte de la position des ses applications relativement aux autres.

Les applications sont listées par ordre de priorités décroissantes. Dans l'exemple de la [Figure 34](#), **testjms** est l'utilisateur privilégié. Ses applications sont donc prioritaires.

Consulter l'état du calendrier

Cela permet d'afficher non seulement, les périodes du calendrier, mais aussi, l'utilisateur associé s'il y en a un.

La date de fin de la période est également affichée (cf. [Figure 35](#)).

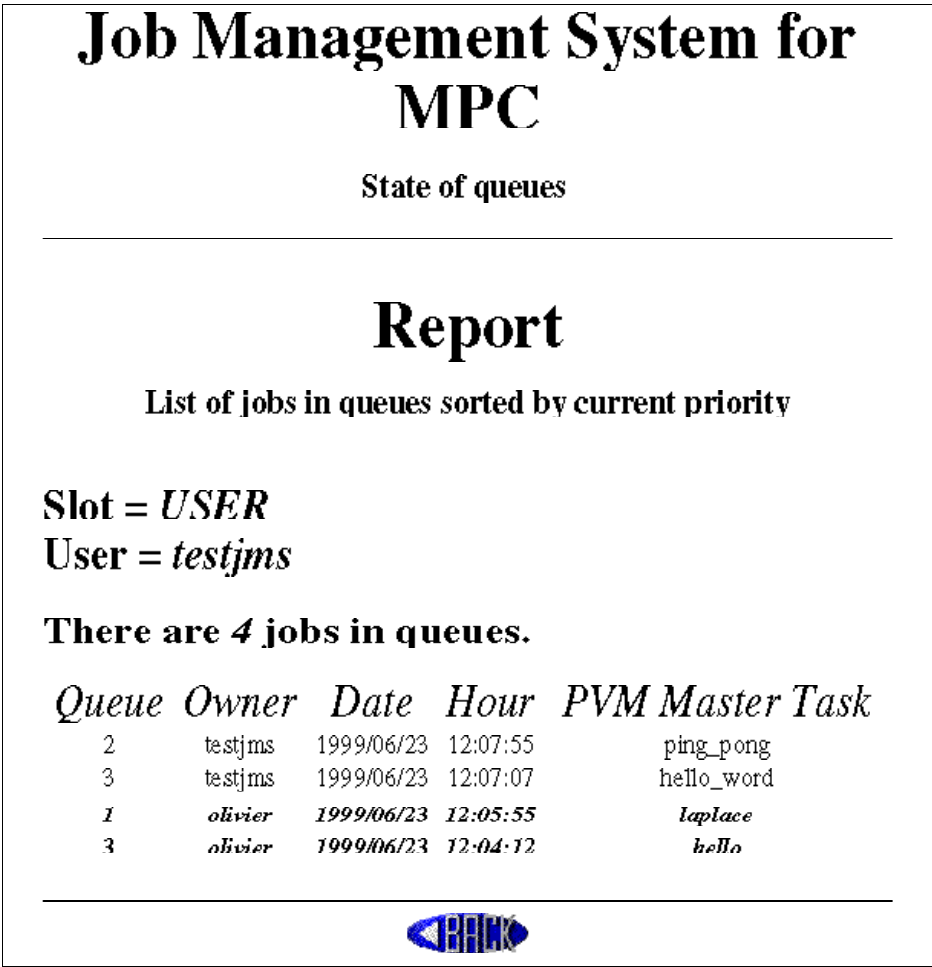


Figure 34 : Etats des files d'attente

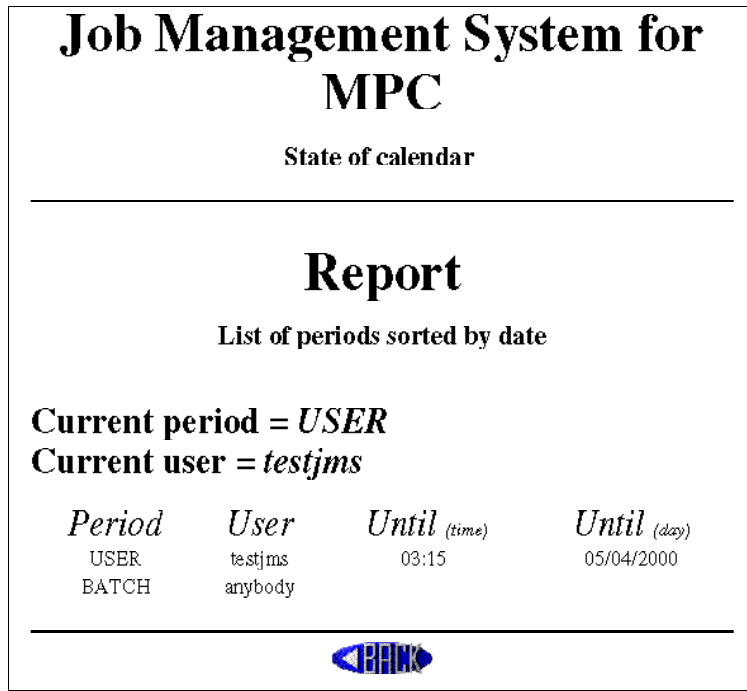


Figure 35 : Etat du calendrier

Devenir l'utilisateur privilégié

Un utilisateur peut, si le calendrier le permet, c'est à dire si la période actuelle est de type USER et qu'aucun utilisateur ne lui est attribué, devenir l'utilisateur privilégié de cette période. Ces applications deviendront alors prioritaires sur celles des autres utilisateurs.

Dans l'exemple de la *Figure 36*, la période du calendrier est bien de type USER mais il y a déjà un utilisateur privilégié. Donc, la demande de l'utilisateur est rejetée.

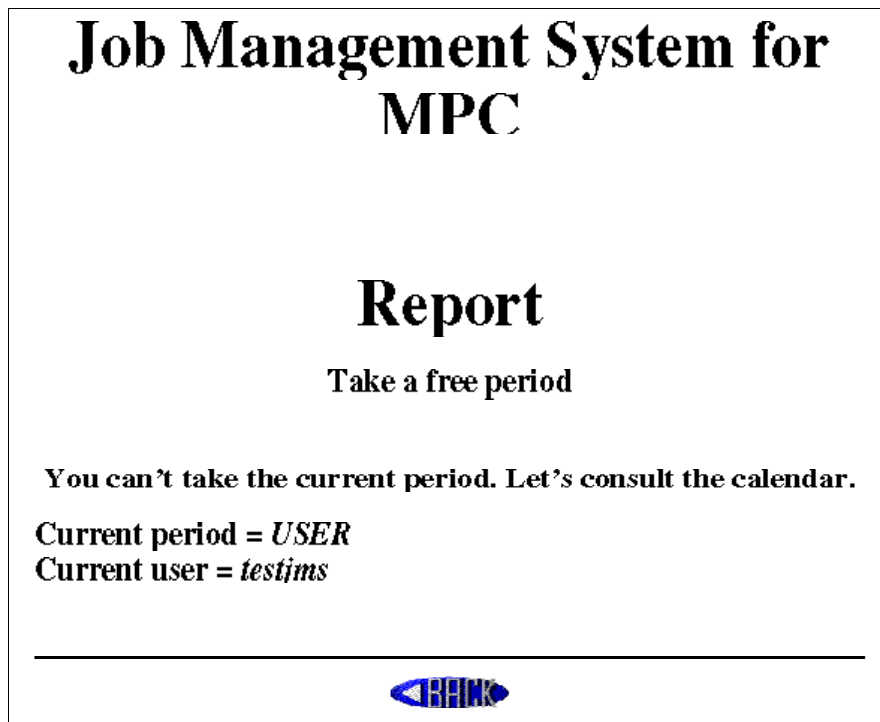


Figure 36 : Devenir l'utilisateur privilégié

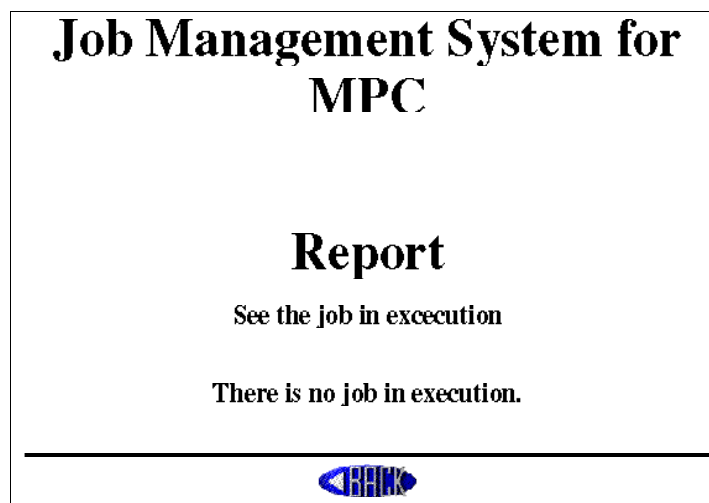



Figure 37 : Voir si une application est en cours d'exécution

Voir l'application qui est en cours d'exécution


L'utilisateur peut voir si une application est en cours d'exécution et, si c'est le cas, consulter les informations relatives à cette application. Dans l'exemple de la *Figure 37*, il n'y avait pas d'application en cours d'exécution.

Tester l'application qui est en cours d'exécution

Un utilisateur ne peut tester qu'une application qui lui appartient et qui est effectivement en cours d'exécution. Pour ce faire, il doit donner un paramètre T en minutes. Le test se fera alors de la manière suivante : toutes les T minutes, le JMS regardera le nombre de communications HSL qui ont eu lieu durant les T dernières minutes. S'il n'y a eu aucune communication, l'utilisateur est averti par email. Ce test s'effectuera aussi longtemps que l'application vivra. Si jamais l'application se termine pour une raison ou pour une autre, le test sera lui aussi arrêté.


LIP6

Asim team
 LIP6 Laboratory
 Paris, France




Job Management System for MPC

Test the job in execution


• **Enter T in minutes :**

If there is no HSL communications **during T minutes** on the node where you have launched your job, you will be informed by an email.

 **Be careful : the communications of your job are going to be test on MPC computer**

Test job

Server design : Olivier Glück
 \$Date: 1999/05/10 15:00:13 \$



Copyright © 1999-2000 UPMC/LIP6
 All rights reserved

Figure 38 : Tester l'application en cours d'exécution

Liste des démons et drivers PVM et MPC

Cela permet de vérifier que tous les démons et drivers sont bien chargés sur tous les nœuds de calcul. Sur chaque nœud, les drivers **cmemdriver**, **hsldriver** et **pvmdriver** doivent être présents ainsi que les démons **hslclient**, **hslserver** et **pvm**. Sinon, aucune application PVM-MPC ne peut s'exécuter correctement.

Attention : les démons sont tués et relancés entre chaque exécution d'une application.

Sur l'exemple de la *Figure 39*, il y a un problème sur tous les nœuds puisque les drivers ne sont pas tous chargés.

```

mpc1-lip6.lip6.fr
DRIVERS : (cmemdriver  hsldriver      pvmdriver)
Type      Id Off Loadaddr Size Info      Rev Module Name
DEV       1 128 f5aaa000 0013 f5aab04c  1 cmemdriver_mod
DEV       2 129 f5cb2000 04d9 f5cc106c  1 hsldriver_mod
DAEMONS : (hslclient  hslserver      pvm    ./vmd)
USER      DTN %CPU %MEM  VSZ  RSS  TT  STAT STARTED  TIME COMMENT

mpc2-lip6.lip6.fr
DRIVERS : (cmemdriver  hsldriver      pvmdriver)
DAEMONS : (hslclient  hslserver      pvm    ./vmd)

mpc3-lip6.lip6.fr
DRIVERS : (cmemdriver  hsldriver      pvmdriver)
DAEMONS : (hslclient  hslserver      pvm    ./vmd)

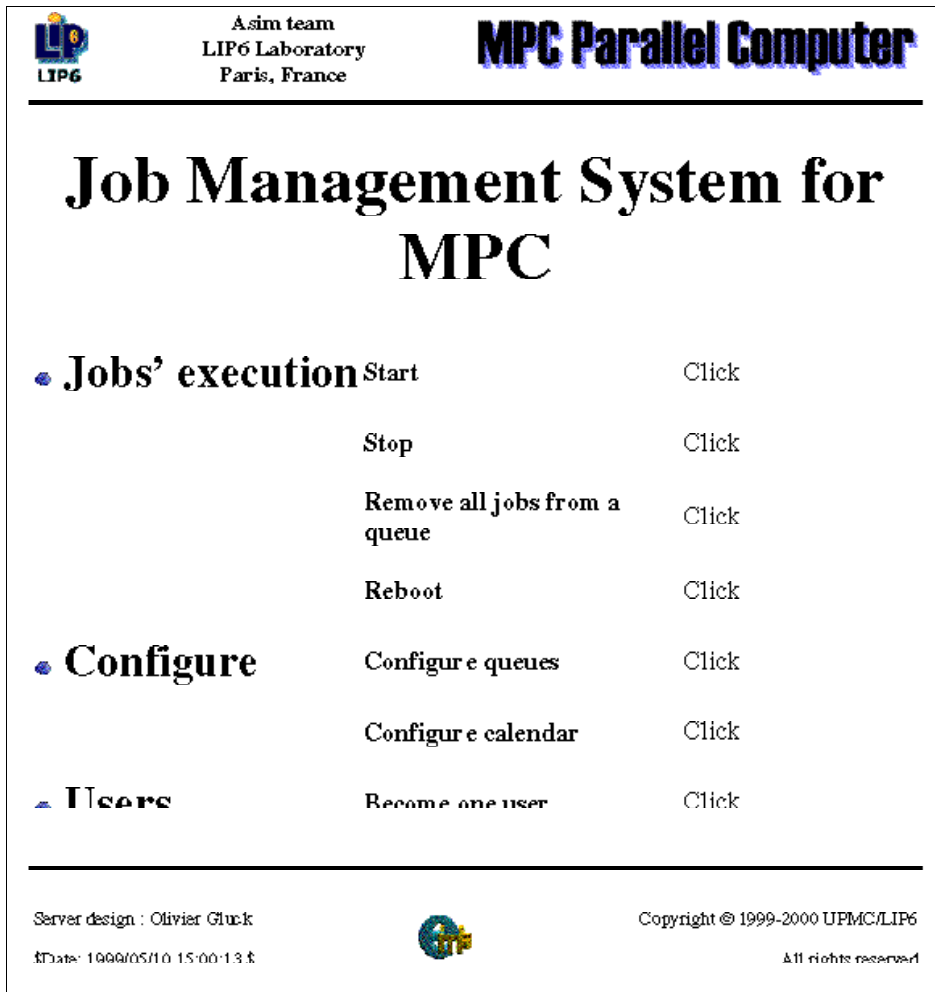
mpc4-lip6.lip6.fr
DRIVERS : (cmemdriver  hsldriver      pvmdriver)
Type      Id Off Loadaddr Size Info      Rev Module Name
DEV       1 128 f5aa2000 0013 f5aa304c  1 cmemdriver_mod
DEV       2 129 f5cc8000 04d9 f5cd706c  1 hsldriver_mod
DAEMONS : (hslclient  hslserver      pvm    ./vmd)
USER      DTN %CPU %MEM  VSZ  RSS  TT  STAT STARTED  TIME COMMENT

```

Figure 39 : Liste des démons et drivers

III.2 Les commandes administrateur

Afin de se connecter à l'interface WWW du JMS, l'administrateur doit saisir son login et son mot de passe. Son login doit obligatoirement être *root*. Il accède ensuite au menu de la [Figure 40](#).



The screenshot shows the administrator interface for the MPC Parallel Computer. At the top, there is a header with the LIP6 logo, the text 'Asim team LIP6 Laboratory Paris, France', and the title 'MPC Parallel Computer'. Below this is the main heading 'Job Management System for MPC'. The interface is divided into three main sections: 'Jobs' execution', 'Configure', and 'Users'. Each section contains several menu items, each with a 'Click' button next to it. At the bottom, there is a footer with server design information, a globe icon, copyright notice, and a date.

Section	Menu Item	Action
Jobs' execution	Start	Click
	Stop	Click
	Remove all jobs from a queue	Click
	Reboot	Click
Configure	Configure queues	Click
	Configure calendar	Click
Users	Become one user	Click

Server design : Olivier Glück
 Date: 10/04/00 15:00:13
 Copyright © 1999-2000 UPMC/LIP6
 All rights reserved

Figure 40 : Menu administrateur

Il dispose alors des fonctions suivantes :

Démarrer l'exécutif

Cela permet le lancement de l'exécution des applications utilisateur PVM-MPC qui sont dans les files d'attente.

Arrêter l'exécutif

Cela permet de stopper le lancement de l'exécution des applications utilisateur PVM-MPC qui sont dans les files d'attente.

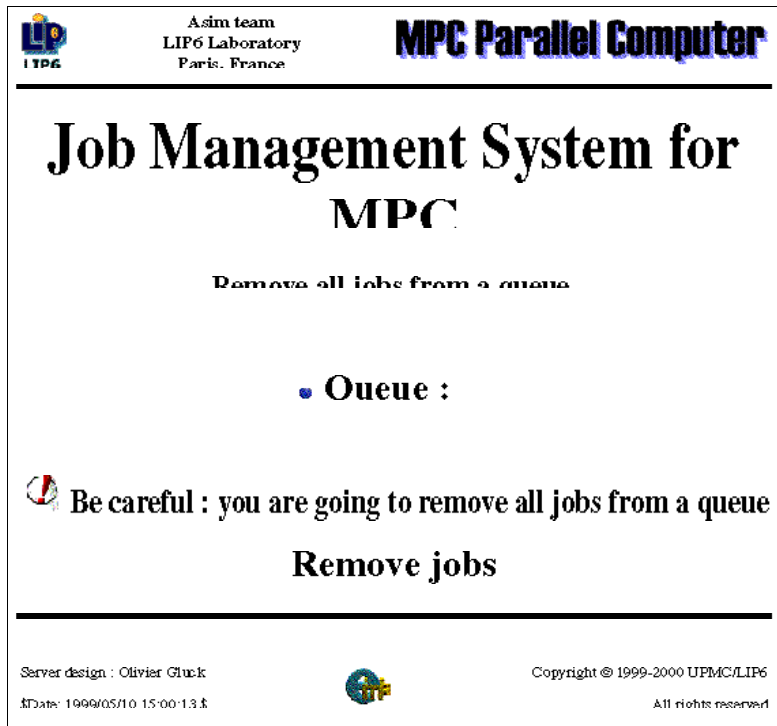


Figure 41 : Vider une file d'attente

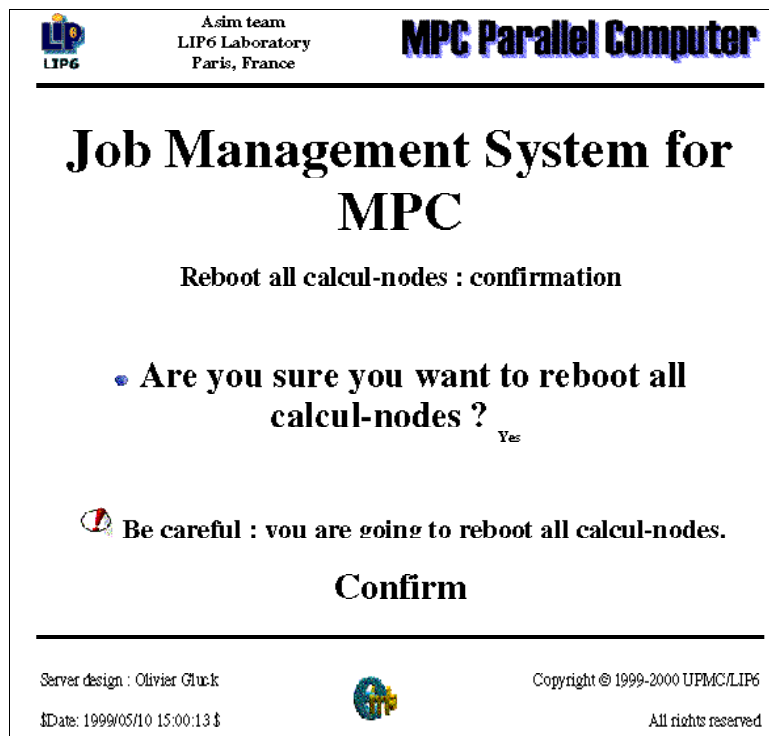


Figure 42 : Redémarrer les nœuds de calcul

Vider une file d'attente

L'administrateur peut effacer tous les fichiers de la file d'attente de son choix. Cela supprimera toutes les applications utilisateur de la file sélectionnée.

Il doit régulièrement vider la queue *old* car tous les fichiers correspondant aux applications dont l'exécution a été lancée sont stockés dans cette file. Le vidage de cette queue consiste à supprimer tous les fichiers exceptés le plus récent. Il est donc déconseillé de vider cette queue « manuellement » en faisant un *rm* dans le répertoire. Il est préférable pour le bon fonctionnement du JMS de réaliser cette opération via l'interface WWW.

Redémarrer les nœuds de calcul

Cette opération redémarre l'ensemble des nœuds de calcul. Mais ce démarrage n'est possible que si le fichier de configuration *config* l'autorise.

Configuration des files d'attente

Cela permet de saisir le paramètre **Tmax** pour chacune des files d'attente. Les valeurs actuelles sont affichées. Tmax (en minutes) correspond au temps maximum d'exécution d'une application issue de la file.

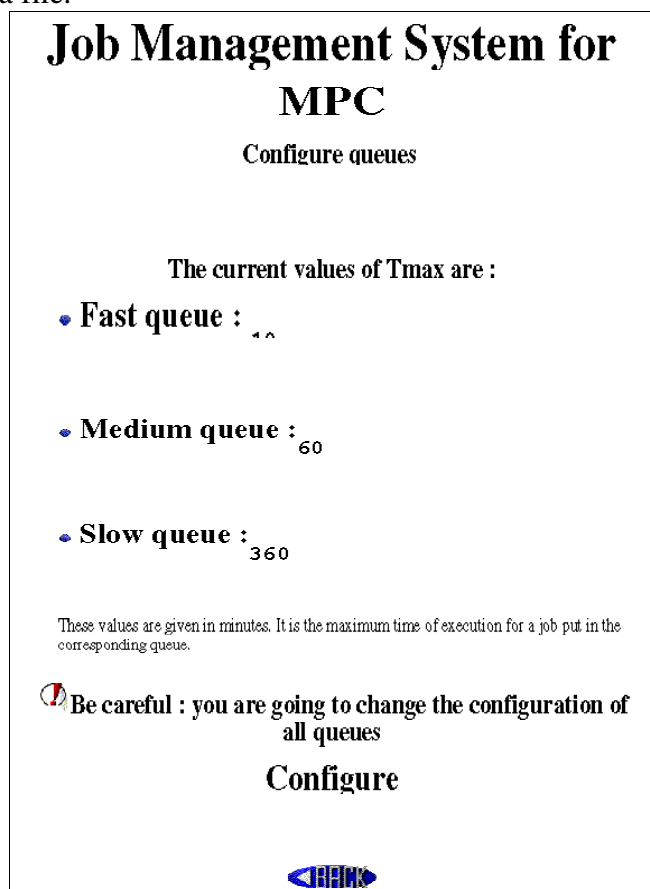



Figure 43 : Configurer les files d'attente



Asim team
LIP6 Laboratory
Paris, France

MPC Parallel Computer

Job Management System for MPC


Configure calendar

- **Enter your display :** `mpc0-lip6.lip6.fr:10.0`

The editor used is vi.


- **Be careful : you are going to edit the configuration file of
calendar**

Edit with vi

Server design : Olivier GlückCopyright © 1999-2000 UPMC/LIP6

\$Date: 1999/05/10 15:00:13 \$All rights reserved

Figure 44 : Configurer le calendrier



Asim team
LIP6 Laboratory
Paris, France

MPC Parallel Computer


Job Management System for MPC

Become one user

- **Login :** `olivier`

- **Be careful : you are going to change your login name**

Change my identity

Server design : Olivier GlückCopyright © 1999-2000 UPMC/LIP6

\$Date: 1999/05/10 15:00:13 \$All rights reserved

Figure 45 : Devenir un utilisateur

Configurer le calendrier

Cela permet d'éditer le fichier calendrier avec l'éditeur *vi*. Il faut pour cela saisir le display pour afficher la fenêtre.

Devenir un utilisateur normal du JMS

L'administrateur peut ainsi exécuter des fonctions du JMS en tant qu'un utilisateur normal. Cela peut par exemple être utile en cas de problèmes ou alors pour accéder aux fonctionnalités utilisateur comme la consultation du calendrier ou des files d'attente.