Université Abou Bekr Belkaid
Tlemcen Algérie
جامعة أبي بكر بلقايد
تلمسان الجزائر

République Algérienne Démocratique et Populaire

Université Abou Bakr Belkaid– Tlemcen
Faculté des Sciences
Département d'Informatique

Mémoire de fin d'études

pour l'obtention du diplôme de Master en Informatique

Option: ***Système d'Information et de Connaissances (S.I.C)***

Thème

Equilibrage de Charge dans Les Systèmes Distribués

Réalisé par :

- **Mr. HENNANE Hemza**
- **Mlle. BELBACHIR Iméne**

Présenté le 25 Juin 2015 devant le jury composé de MM.

- *Mr.MERZOUG Mohamed* ***(Président)***
- *Mr. SMAHI Mohamed Ismail* ***(Encadreur)***
- *Mr.BOUAFIA Zouheyr* ***(Co-Encadrant)***
- *Mr. ETCHIALI Abdelhak* ***(Examinateur)***
- *Mr. ABDELAOUI Ghouti* ***(Examinateur)***

Année Universitaire: 2014-2015

REMERCIEMENTS

En préambule nous tenons remerciant ALLAH qui nous aide et nous donne la patience et le courage durant cette année.

Nous tenons à remercier tous ceux qui nous ont aidés, d'une manière ou d'une autre, pendant ce travail d'étude et de recherche.

Nous tenons d'abord à remercier très chaleureusement Monsieur **SMAHI Ismail** qui nous a permis de bénéficier de son encadrement. Les conseils qu'il nous a prodigué, la patience, la confiance qu'il nous a témoignés ont été déterminants dans la réalisation de notre travail de recherche.

Nos remerciements à Monsieur **BOUAFIA Zouheyr** pour ses précieux conseils, la rectification ainsi que la validation de notre plan et pour sa disponibilité.

Nos vifs remerciements vont également aux membres du jury pour l'intérêt qu'ils ont porté à notre recherche en acceptant d'examiner notre modeste travail Et de l'enrichir par leurs propositions.

Nos remerciements s'étendent également à tous nos enseignants durant les années des études.

Merci à Tous et à Toutes.

SOMMAIRE

Introduction Générale

Devant les limites physiques de la taille de la mémoire et de la vitesse de calcul des systèmes monoprocesseurs, les systèmes parallèles et distribués apparaissent comme une solution compétitive pouvant résoudre les problèmes nécessitant de grandes capacités de calcul et la manipulation de grandes quantités de données.
Plusieurs avantages caractérisent ces systèmes tels que leur grande disponibilité, leur meilleure tolérance aux pannes, leur grande flexibilité et surtout, ils permettent de partager de nombreuses ressources (processeur, mémoire, disque, etc.) à travers un réseau.

L'utilisation optimale des ressources de calcul des systèmes distribués est un aspect très important qui mobilise beaucoup de chercheurs à travers le monde. Cette optimalité nécessite une répartition de la charge de travail sur les différents nœuds de calcul.
Il faut en effet éviter, dans la mesure du possible, les situations où certains nœuds sont surchargés alors que d'autres sont sous-chargés ou complètement libres. L'équilibrage de la charge de travail sur les diverses ressources disponibles s'avère être un véritable défi.
Le but de ce projet est d'étudier les différents systèmes parallèles et distribués existants jusqu'à maintenant. Après une synthèse des algorithmes d'équilibrage de charge dans les systèmes distribués, nous proposons un modèle d'échange des messages entre deux ordinateurs à travers un réseau local LAN. Chaque ordinateur collecte son information de charge (Processeur, Mémoire) et l'envoie à l'autre ordinateur par l'utilisation de la bibliothèque MPI.

Le reste de ce mémoire est organisé comme suit :

Dans le premier chapitre, nous donnons un bref aperçu sur les différents systèmes distribués à savoir Les Grappes Informatiques, les Grilles informatiques et enfin l'informatique en Nuages (Clouds), ensuite nous présentons la classification de ces systèmes.

Dans le deuxième chapitre, nous présentons un échantillon d'algorithmes d'équilibrage de charge pour chaque type de système distribué avec un exemple de déroulement.

Dans le troisième chapitre, nous présentons la programmation parallèle en présentant ses avantages, nous nous intéressons aux modèles de programmation parallèle et plus particulièrement au modèle à échange de messages (MPI).

Dans le chapitre 4, nous détaillerons notre travail qui est une communication entre deux ordinateurs dans un réseau local, où chaque ordinateur collecte son information de charge (Processeur, Mémoire) et l'envoie à l'autre ordinateur.

Enfin, nous terminons ce mémoire par une conclusion qui résume notre travail et par une proposition de quelques perspectives de recherche.

Introduction

Alors que l'Internet et les technologies de réseau ont progressé dans la sophistication et la fiabilité, les ingénieurs ont créé un nouveau mode de gestion des services informatiques : sont **Les Systèmes Distribués**. Au lieu de centraliser les données et la puissance de calcul dans un seul endroit, puis de l'envoyer aux clients, les systèmes distribués répartis les données et les tâches de calcul sur plusieurs nœuds qui fonctionnent à l'unisson. Bien que ce type de système présente de nombreux avantages comme **[1] :**

- Transparence (Masquer la répartition) : La séparation physique entre machines et les différences matériels/logiciels pour les accès sont invisibles par l'utilisateur
- Migration des ressources possible sans interférence avec la localisation physique (ex. transférer un objet uniquement par son nom logique sans modification de ce nom et sans modification de l'environnement d'un utilisateur);
- Réplication de ressources non visible ;
- Concurrence d'accès aux ressources non perceptible (ex. accès à un même fichier ou une table dans une base de données : mécanisme de verrou ou de transaction) ;
- La répartition la charge des calculs et des données entre les différents nœuds ;
- Tolérance aux pannes permettant à un utilisateur de ne pas s'interrompre (ou même se rendre compte) ;
- Flexibilité (facilité d'utilisation et de configuration) ;
- Extensibilité (Ajout/MAJ de composants sans en affecter les autres) ;
- Mise à l'échelle (Scalability) : fonctionne efficacement dans différentes échelles.

1) Définition

Selon ***Tanenbaum*** une définition d'un système distribué au sens large serait la suivante **[2]** :

Un système distribué est un ensemble d'ordinateurs indépendants qui apparaît pour ses utilisateurs comme un seul système cohérent.

Plus précisément, on peut voir un système distribué comme un ensemble d'unités de calcul (appelé Nœud). Possédant les caractéristiques suivantes :

- Chaque nœud possède son propre espace d'adressage ;
- Le nombre de nœud dans le système distribué est arbitraire ;
- Le système est capable de traiter un nombre quelconque de processus ;
- La communication entre les différents nœuds se fait au moyen de messages. Il convient, dès lors, de tenir compte du délai de communication entre les différents nœuds ;
- Les interactions entre les différents processus se font de manière coopérative, et non sur base du modèle maître/esclave ;
- En cas de panne d'une ou plusieurs unités de calcul, le système doit pouvoir se reconfigurer (la notion de Tolérance au panne).

Remarquons que tous les auteurs n'adhèrent pas à l'ensemble de ces caractéristiques et plus particulièrement aux deux dernières.

2) Les Différent Systèmes Distribués Existants

Nous présentons ici trois grands types : les Fermes de calcul ou clusters, les Grilles de calcul et les Clouds dont le paradigme sous-jacent est l'informatique en nuage **[3].**

2-1 Les clusters ou fermes de calcul :

Les scientifiques ont besoin d'effectuer des calculs de plus en plus massifs dans leur travail de recherche ; c'est ce qui a poussé le développement d'ordinateurs toujours plus puissants : les superordinateurs. Dans les deux dernières décennies, afin de satisfaire cette demande en superordinateurs, les industriels ont regroupé des ordinateurs afin d'accroître les capacités de calcul : ce sont les fermes de calcul (cluster). Plutôt que fabriquer un superordinateur avec un processeur très puissant, les industriels préfèrent concevoir des ordinateurs sur la base de processeurs existants afin de constituer une ferme de calcul. Avec ce procédé de tels superordinateurs reviennent moins cher à fabriquer.

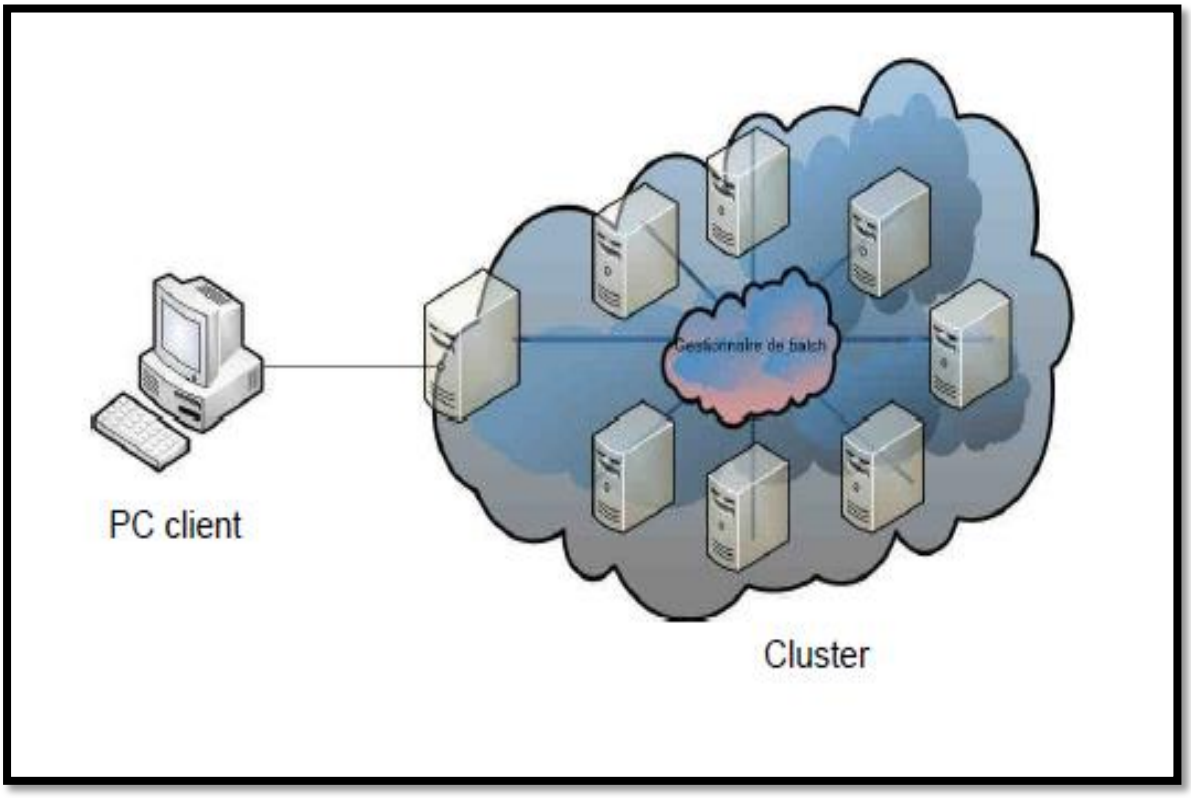
Une ferme de calcul, est constituée de différents serveurs composés d'un ou de plusieurs processeurs. Ces serveurs sont connectés au moyen de réseaux de communication très rapides. L'utilisateur se connecte à la ferme de calcul sur un serveur frontal et exécute son application parallèle qui est conçue pour s'exécuter en parallèle sur un ensemble de processeurs.

Généralement les serveurs qui composent une ferme de calcul sont tous similaires : ce sont des plates-formes Homogènes. De plus, les serveurs d'une ferme de calcul sont physiquement localisés au même endroit.

De nos jours les fermes de calcul sont très répandues dans le monde scientifique, dans les universités, les centres de recherche pour les calculs hautes performances (HPC). Elles sont utilisées pour effectuer des simulations de tout genre : simulations physiques (aérodynamique, nucléaire, et autres), prévisions météorologiques (superordinateur K de Fujitsu), Application à la médecine ou encore à la génétique (Blue Gene).

2-1-1 Définition

Un cluster, en français Grappe, est une concentration de deux ou plusieurs machines, qui accèdent à des données communes **[4].**



.

Figure I-1 : **Présentation générale d'un cluster**

2-1-2 L'architecture d'un cluster

Il existe différentes architectures de cluster qui se distingue par leurs manières d'accéder au domaine de stockage.

A / Principe de partage

Un cluster peut adopter l'un des stratégies de partage suivantes :

- Soit on permet à tous les nœuds d'accéder au même domaine de stockage. Des problèmes de concurrence se posent alors. Il faut gérer les accès parallèles en écriture.
- Soit on exclu l'accès commun aux mêmes données.

B/ Principe du Mirroring

Avec le principe de Mirroring, chaque nœud est disponible sur un domaine de stockage, reflété sur une connexion de réseau dédiée.

C/ La Technologie du RAID

Elle permet de constituer une unité de stockage à partir de plusieurs disques durs , Ou bien une plus grande capacité/vitesse d'écriture (suivant le niveau du *RAID* choisi).

La répartition des données sur plusieurs disques durs permet donc d'en augmenter la sécurité, de contrôler le coût et de fiabiliser les services associés.

2-1-3 Solutions d'Interconnexion

L'obtention de performances élevées lors de l'exécution d'applications sur les clusters nécessite de réduire au maximum le coût des communications entre les différentes machines.

A cet effet, plusieurs solutions réseaux sont disponibles plus ou moins adaptées à une utilisation dans un environnement de Grappes **[5].**

A/ Ethernet (100 BaseT et GigaByte Ethernet) : C'est le Protocole de connexion de Réseaux locaux le plus utilisée dans le monde, mais reste mal adaptée pour un usage dans un environnement réparti à haute performance, et ce pour plusieurs raison parmi lesquelles :

- Latence trop Importante.
- Bande passante relativement faible.
- Support logiciel beaucoup trop lourd avec copies successives des messages ce qui dégrade fortement les performances système.
- Forte consommation énergétique des adaptateurs réseau.

B/ Myrinet : Myrinet est le leader du marché des réseaux d'interconnexion pour Grappes de Calcul.

Ses principales caractéristiques sont :

- Contrôle de flux, d'erreurs, ainsi que l'état des liens en continu ;
- Excellent passage à l'échelle ;
- Faible latence, offrant une grande disponibilité aux applications ;
- Architecture commutée (commutateur peut relier jusqu'à 256 nœuds) ;
- Services *« PIO & OS By-pass »* pour s'affranchir du système d'exploitation pour l'émission et la réception des messages en effectuant directement les opérations de lecture et d'écriture, ce qui permet une amélioration des performances.

C/ InfiniBand

InfiniBand est une norme créée à la fin des Années **1990** par un consortium de nombreux constructeurs de matériel informatique afin de définir l'Architecture des entrées sorties de l'avenir.

La norme décrit une Architecture et des spécifications pour les flux de données entre processeurs et composants d'E/S intelligents. *InfiniBand* est destinée à remplacer le bus PCI dans les serveurs et offrir une meilleure capacité, à augmenter la capacité d'extension et à améliorer la souplesse de conception des serveurs. Elle permet essentiellement de relier des serveurs, des systèmes de stockage distants et d'autres composants de réseau dans une matrice contenant des commutateurs et des liens. L'Architecture basée sur les commutateurs peut connecter jusqu'à **64000** serveurs, systèmes de stockage et composants de réseau.

InfiniBand a construit une architecture de communication système, basée sur le concept de canal ; Avec cette Architecture, il n'est plus nécessaire d'avoir le matériel d'interface d'E/S dans le châssis du serveur : le stockage distant, l'utilisation des réseaux et les connexions entre les serveurs s'effectuent en reliant tous les périphériques à une matrice centrale de commutateurs et de liens. En enlevant les E/S du châssis du serveur, on en augmente la densité, ce qui permet d'envisager un centre des données plus souple et évolutif puisque l'on peut ajouter des nœuds indépendants à la demande.

2-2 Les grilles informatiques

Une Grille Informatique **(Grid Computing)** peut être considérée comme un superordinateur virtuel qui est physiquement constitué de plusieurs machines différentes et délocalisées. Cette infrastructure est dite hétérogène car les éléments qui la constituent ont des vitesses de calcul différentes et des capacités de stockage différentes. Les éléments peuvent aller d'une ferme de calcul à un ordinateur personnel. Ces machines sont interconnectées par des réseaux de communication beaucoup moins rapides que ceux d'une ferme de calcul. Elles sont par exemple interconnectées au moyen de câbles Ethernet ou du wifi. Les Grilles sont soit privées ou soit publiques et dans ce cas elles sont accessibles à tous depuis internet.

Les grilles informatiques peuvent être classées dans trois grandes familles :

- les Grilles pair-à-pair (*Peer-To-Peer Grids* ou Desktop Grids) sont constituées d'une multitude de machines de petite taille, comme des ordinateurs personnels. Le problème à résoudre est découpé en sous problèmes résolus indépendamment sur les machines, avant qu'une machine maître ne récolte les différents résultats.
- les Grilles de ressources sont constituées de machines de plus grande taille, telles que des fermes de calcul, comme Grid'50001.
- les Grilles de services proposent aux utilisateurs des applications déployées sur la grille. Ce sont des architectures orientées services (Service Oriented Architecture ou SOA). Par exemple l'utilisateur fait la demande d'une application à la grille. En cas de disponibilité l'utilisateur fournit ses données à l'application puis récupère ses résultats de manière transparente sans se soucier, ni de la localisation du service, ni de la manière dont le service est réalisé.

2-2-1 Définition

Une grille informatique (en Anglais, Grid) est une infrastructure virtuelle constituée d'un ensemble de ressources informatiques potentiellement partagées, distribuées, hétérogènes, délocalisées et autonomes.

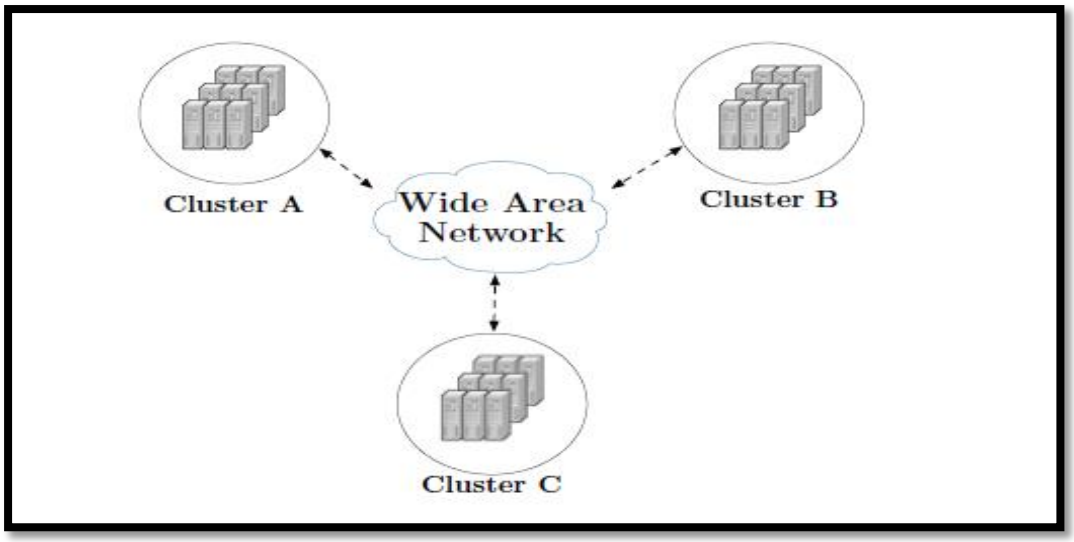


Figure I-2 : **Schéma générale d'une grille informatique**

2-2-2 L'architecture d'une grille de calcul

L'architecture d'une grille est organisée en couches. Elles seront énumérées dans un premier temps et décrites ensuite. Une couche est une abstraction représentant un ensemble de fonctions du système qui permettent des actions de même niveau. Chaque couche fait appel aux services de n'importe quelle couche inférieure **[6].**

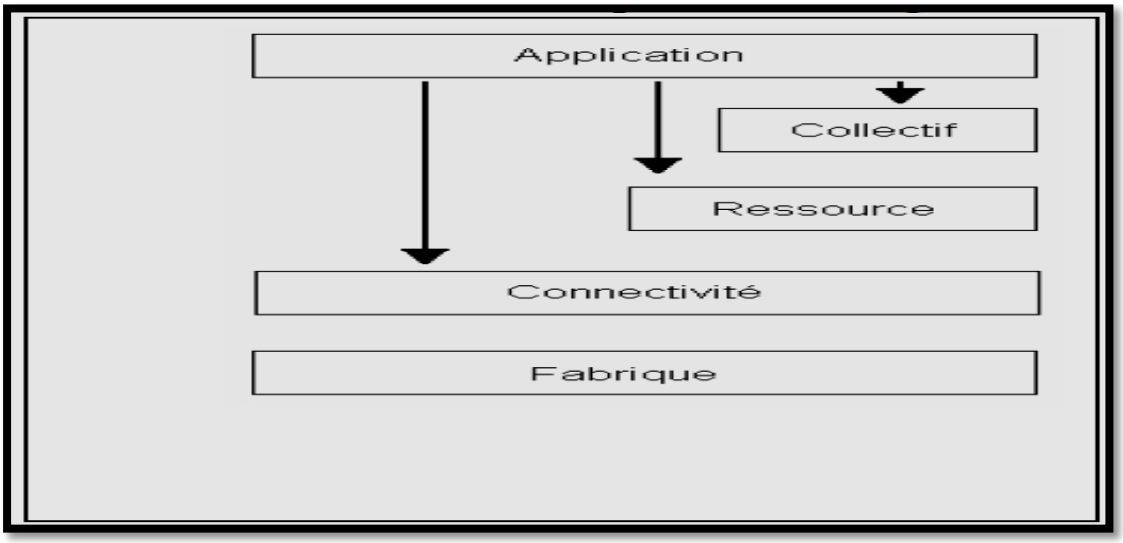


Figure I-3 : **Le modèle de l'architecture en couches**

La couche application : représente l'ensemble des fonctions qui ont été développées pour interagir avec la grille.

La couche collective : se charge de la coordination des ressources. Elle comprend les services d'annuaire, la prise en charge des demandes, la surveillance et la réplication des données. C'est l'orchestrateur des ressources.

La couche ressource : représente le partage de ressources individuelles. C'est elle qui a notamment en charge la mise à disposition et la facturation à l'utilisation.

La couche connectivité représente les protocoles fondamentaux régissant la communication et l'authentification.

La couche fabrique : représente le niveau le plus proche des ressources 'physiques'. Il s'agit de l'ensemble des serveurs, le stockage, et le réseau qui fournissent les ressources nécessaires aux applications.

L'architecture d'une grille de calculs est parfois représentée sous une forme de sablier. La métaphore du goulet d'étranglement au *niveau Connectivité* permet d'imager l'idée de la limitation du nombre de protocoles utilisés (protocoles de communication et protocoles réseau) pour faciliter le partage des ressources. A l'inverse, la base et le haut du sablier (les parties élargies) symbolisent la diversité des ressources et des applications.

L'utilisation de protocoles standards simplifie également le développement et le déploiement des solutions. Ainsi, pour la communication et la sécurité, de nombreux protocoles développés pour Internet ont été réutilisés. Pour la communication, les protocoles nécessaires sont ceux relatifs au transport, au routage et à la gestion des noms. Le choix se porte essentiellement sur les protocoles de la pile TCP/IP. TCP et UDP sont utilisés pour le transport, IP pour le routage, ICMP pour la surveillance, et pour les applications est utilisé DNS.

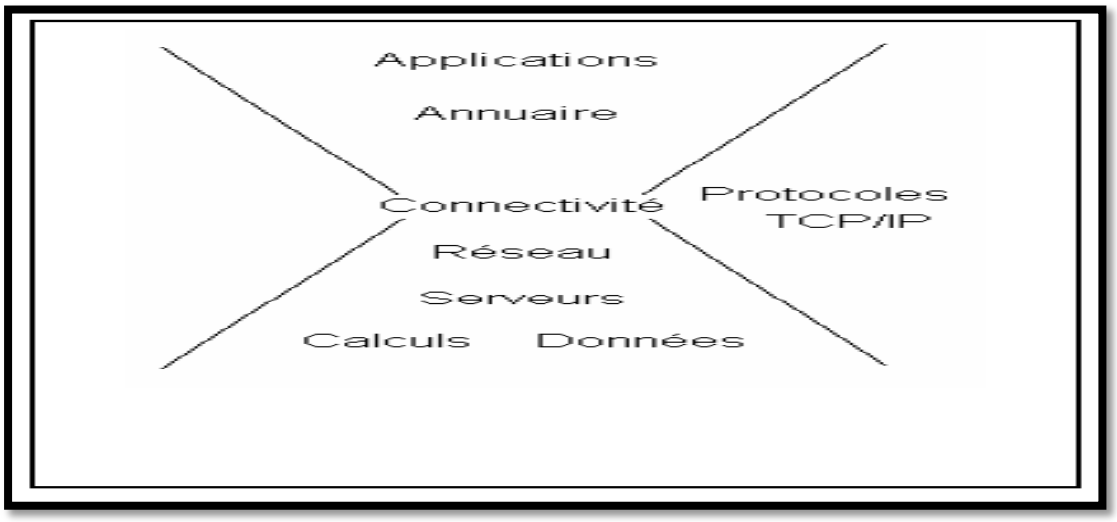


Figure I-4 : **Le modèle de l'architecture en sablier**

2-2-3 Les différentes topologies de grilles

Énumère les grilles d'un point de vue topologique en trois types par ordre croissant d'étendue géographique et de complexité [**7]** :

- ✓ ***Intra-grappe*** (en analogie avec Intranet) : la plus simple des grilles est l'intra-grappe, composée d'un ensemble relativement limitée de ressources et de services et appartenant à une organisation unique. Les principales caractéristiques d'une telle grille sont l'interconnexion à travers un réseau performant et haut débit, un domaine de sécurité unique et maîtrisé par les administrateurs de l'organisation et un ensemble relativement statique et homogène de ressources.
- ✓ ***Extra-grappe*** (en analogie avec Extranet) : une extra-grappe étend le modèle en regroupant plusieurs intra-grappe. Les principales caractéristiques d'une telle grille sont la présence d'un réseau d'interconnexion hétérogène haut et bas débit (LAN/WAN), de plusieurs domaines de sécurité distincts, et d'un ensemble plus ou moins dynamique de ressources. Un exemple d'utilisation est lors d'alliances et d'échanges « Business -to- Business » (B2B) entre entreprises partenaires.
- ✓ ***Inter-grappe*** (en analogie avec Internet) : une inter-grappe consiste à agréger les grilles de multiples organisations, en une seule grille. Les principales caractéristiques d'une telle grille sont la présence d'un réseau d'interconnexion très hétérogène haut et bas débit (LAN / WAN), de plusieurs domaines de sécurité distincts et ayant parfois des politiques de sécurité différentes et même contradictoires, et d'un ensemble très dynamique de ressources.

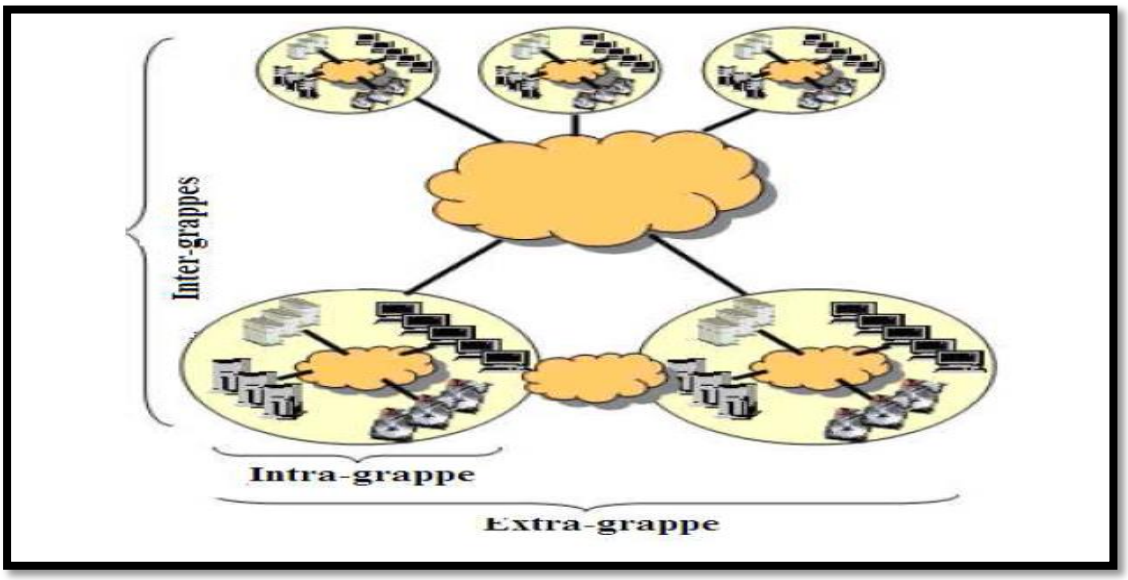


Figure I-5 **: Les différentes topologies d'une grille de calcul**

2-3 L'informatique en nuage

L'informatique en nuage (Cloud Computing) est un paradigme qui propose à l'utilisateur de délocaliser ses ressources de calcul et ses ressources de stockage. Les ressources qui servent de support au Cloud Computing sont généralement des clusters car ceux-ci sont plus homogènes et disponibles que les grilles de calcul. À la manière d'un réseau électrique, les entreprises paient la consommation réelle des services proposés par le cloud du fournisseur qui leur garantit une qualité de calcul et de stockage. Avec ce concept, elles n'ont plus besoin de se préoccuper de l'achat de serveurs de calcul et de stockage ou de l'organisation des infrastructures réseaux.

2-3-1 Définition

L'informatique en Nuage" est un terme assez vague possédant différentes significations, qu'elles soient extrêmement ciblées ou plus larges jusqu'à englober l'internet.

L'US National Institute for Standards and Technology a fourni l'une des définitions les plus claires et les plus communément admises **[8]** :

" L'informatique en nuage est un modèle pratique, à la demande et universel, permettant d'établir un accès par le réseau à un réservoir partagé de ressources informatiques configurables (par exemple, réseaux, serveurs, stockage, applications et services) qui peuvent être rapidement mobilisées et mises à disposition en minimisant les efforts de gestion et les contacts avec le fournisseur de services"

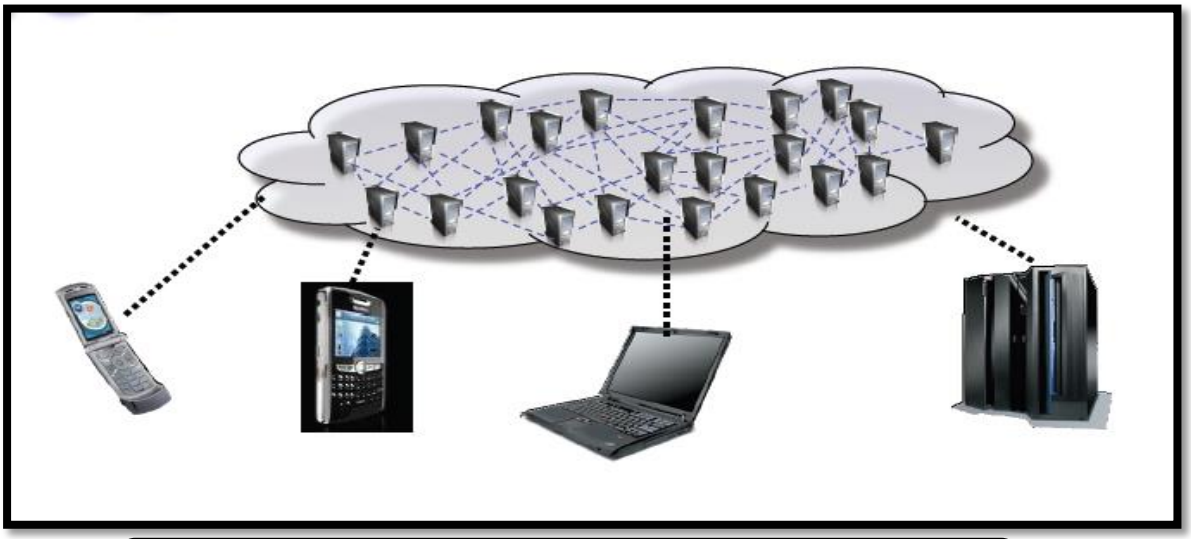


Figure I-6 : **Schéma générale des clouds (Les nuages)**

Le Cloud Computing consiste à déporter sur des serveurs distants des données et des traitements informatiques traditionnellement localisés sur des serveurs locaux, voire sur le poste de l'utilisateur. Il permet l'accès via un réseau, généralement entendu comme Internet, à la demande et en libre service, à des ressources informatiques virtualisées et mutualisées habituellement facturées à l'usage **[9]**.

2-3-2 Les Caractéristique de Cloud Computing [10]

- **Elasticité :** Les utilisateurs de Cloud Computing à des capacités illimitées, qui peuvent être augmentées ou réduites selon l'usage.
- **Pay-As-you-Use :** Les clients du Cloud vont donc recevoir des tarifications les plus précises possibles, pour les ressources qu'ils utilisent.
- **Self-Service :** Capacité à fournir une ressource informatique automatiquement, sans requérir d'interaction humaine côté fournisseur.
- **Mesure de la qualité de services :** Evaluer et garantir un niveau de performance et de disponibilité adapté aux besoins spécifiques des clients.
- **Accès réseau universel:** L'accès très rapide des ressources à laide d'un réseau, par des protocoles standards en manière très élasticité.
- **Mise en commun de ressources :** Datacenter fournissant les ressources (machines, stockage, etc.) pour les différentes clients en monde partagé.
- **Multi-Tenancy :** La capacité de fournir un service simultanément à plusieurs clients, cela permet également d'augmenter l'utilisation des ressources déployées.
- **Disponibilité :** La haute disponibilité de la plate-forme est obtenue par redondance et par la capacité de se remettre rapidement en cas des problèmes.
- **Green IT :** ce concept qui désigne l'ensemble des nouvelles technologies à faible impact environnemental. Le Cloud Computing, basé sur la virtualisation, qu'il réduise les risques des machines physiques sur l'environnement et la consommation énergétique.

2-3-3 Le fonctionnement de l'Informatique en nuage

Le nuage est un modèle issu de l'infrastructure informatique mondiale développée par de grandes entreprises comme Google, Amazon, Microsoft et eBay, qui l'utilisaient initialement dans le cadre de leurs propres activités. À la suite de l'établissement de centres de données de grande taille dans plusieurs pays dotés de connexions à très haut débit à l'internet mondial, Ces entreprises ont décelé une source de revenus potentiels consistant à fournir à d'autres entreprises leurs services informatiques et leurs capacités de stockage de données excédentaires. Certains de ces centres de données peuvent accueillir jusqu'à **100 000** serveurs.

Chacun de ces serveurs informatiques exécute un système d'exploitation qui est capable de mettre plusieurs environnements "virtualités" à disposition de clients, qui peuvent y exécuter leurs propres applications logicielles sans perturber l'exécution simultanée d'autres programmes sur le même serveur. Certaines entreprises comme IBM, HP et Citrix commercialisent des systèmes qui gèrent de façon efficace ce processus de virtualisation et qui offrent des fonctionnalités supplémentaires garantissant la fiabilité et la sécurité. Par exemple, dans les systèmes "CloudSystem Matrix d'HP", les programmes sont automatiquement transférés vers une machine qui peut les prendre en charge lorsqu'une autre machine tombe en panne. Les centres de données peuvent également stocker des données de clients sur demande. Les clients peuvent demander à tout moment des ressources informatiques et de stockage en fonction de leurs besoins, sur la base d'un paiement à l'usage.

2-3-4 Les différents Types de Nuages

Les services d'informatique en nuage les plus connus sont les **"Nuages Publics",** qui sont fondés sur des réseaux mondiaux de centres de données *(Datacenter)* offrant des services basés sur un modèle de paiement à l'usage et destinés au grand public ou à un grand groupe industriel. Grâce à l'accumulation de la demande, à l'achat groupé de matériel informatique et d'énergie et à la réduction des coûts unitaires de la main-d'œuvre, les économies de coûts les plus élevées sont concentrées à ce stade. Les fournisseurs de services d'informatique en nuage offrent même certains services gratuitement pour attirer des clients.

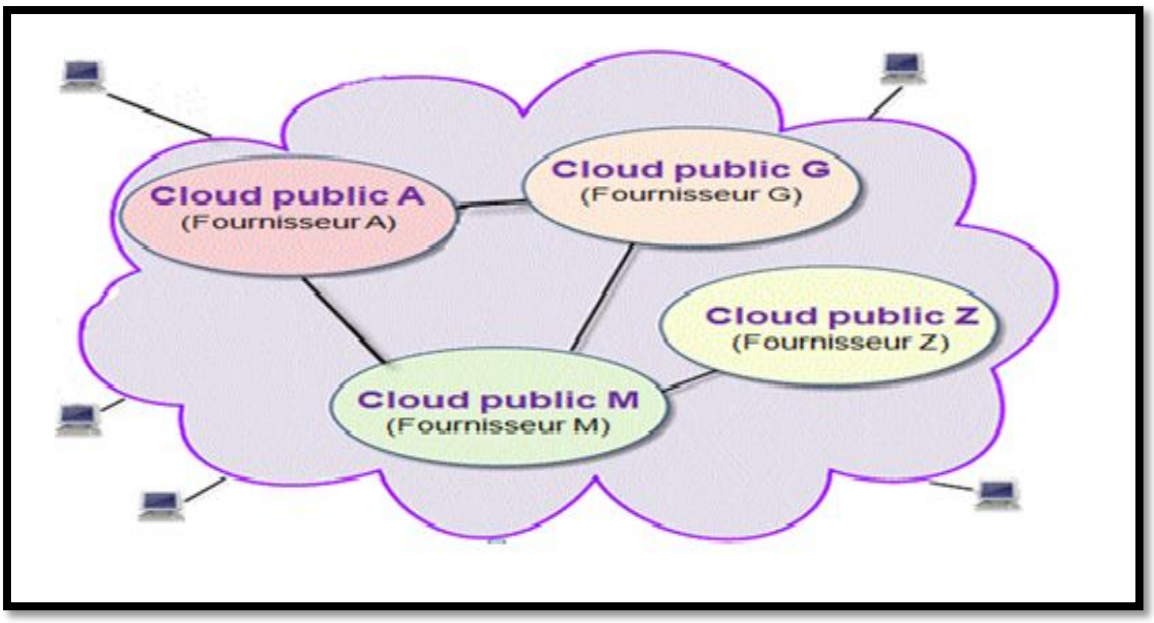


Figure I-7 : **Cloud public**

Les Entreprises et les administrations peuvent concevoir leurs propres **"Nuages Privés"** à partir de leur matériel informatique. Il s'agit souvent de la première étape de transition des systèmes informatiques existants vers des services d'informatique en nuage public. Même si les nuages privés n'offrent en général pas autant de gains de coûts que les nuages publics, ils peuvent être adaptés aux besoins de certaines organisations qui possèdent des données sensibles qu'elles ne veulent pas transférer hors de leurs systèmes.

Cette différence de coûts notable entre les différents types de nuages est illustrée par le modèle de calcul des coûts du nuage de Microsoft, qui révèle que les nuages privés sont extrêmement coûteux lorsqu'ils comptent moins de **100** serveurs puisqu'ils n'offrent aucun des avantages liés aux économies d'échelle pour l'offre et la demande des nuages publics. Lorsqu'il abrite environ **1 000** serveurs, un nuage privé offre des avantages plus importants même si l'on rapporte que ceux-ci restent toujours dix fois inférieurs par rapport aux services d'informatique en nuage public. Les entreprises doivent également investir préalablement dans les équipements matériels et logiciels nécessaires.

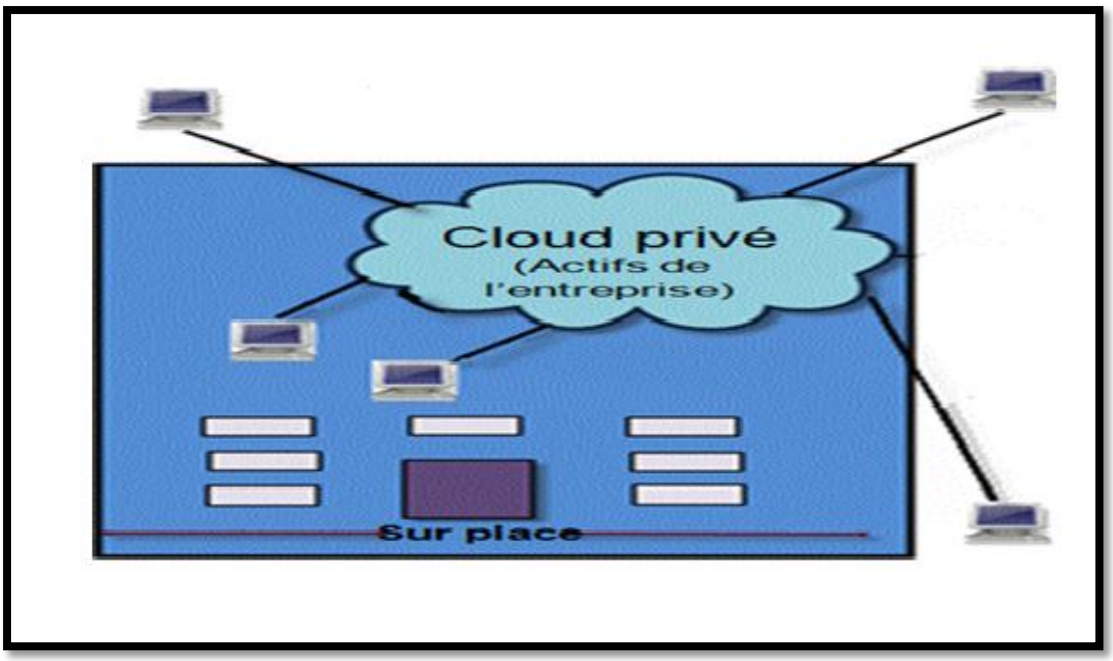


Figure I- 8 : **Cloud privé**

Le **"Nuage Hybride"** est une solution intermédiaire entre le nuage public et le nuage privé. Les ressources de calcul et de stockage sont partagées entre le nuage public et les systèmes privés. Cela permet de traiter les données plus sensibles en interne et les autres données sur des serveurs de nuage public moins coûteux.

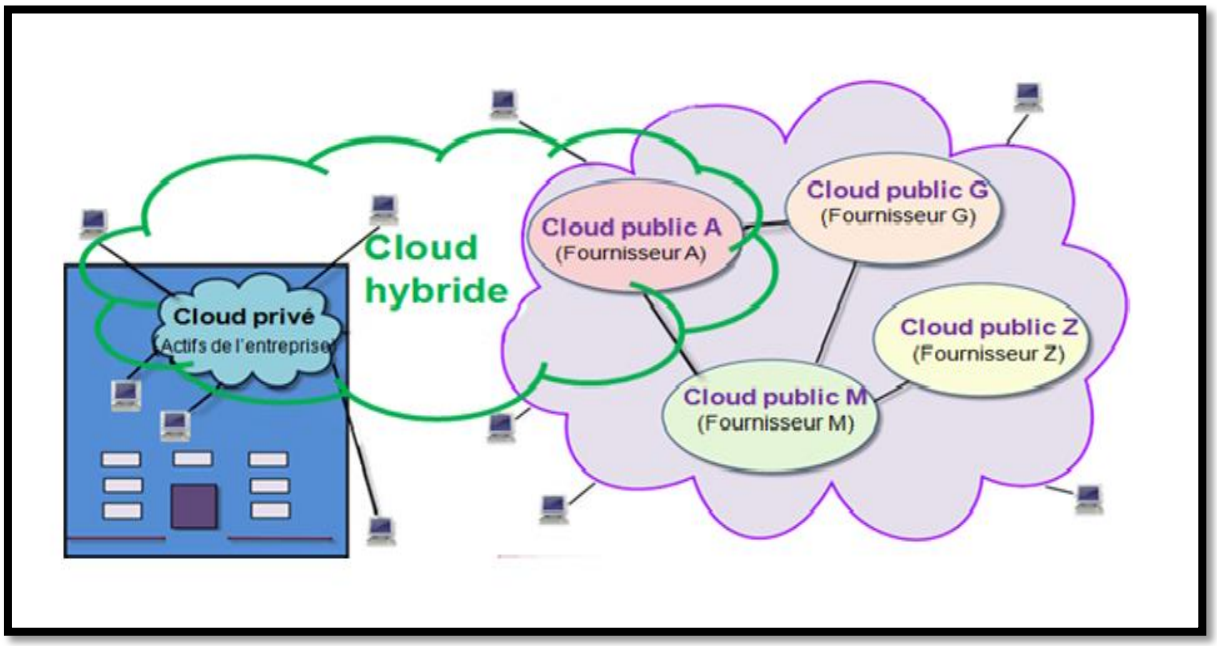


Figure I-9 : **Cloud hybride**

Le **"Nuage Communautaire"** est un autre type de nuage créé sur mesure et exploité par un groupe d'organisations ayant convenu de règles communes de sécurité et de respect de la vie privée et d'autres règles relatives à ce nuage. La demande et les économies de coûts potentielles dépendent, entre autres facteurs, de la taille et du nombre d'organisations concernées.

Figure I-10 : **Cloud community**

2-3-5 Classification des services d'informatique en nuage

Les systèmes d'informatique en nuage sont en général classés par catégories de services. Les principaux types de services sont les suivants :

- **Stockage en tant que service :** Ce service permet aux clients de stocker et de partager leurs données à distance. Exemples : *Dropbox, Box.net, Amazon Scalable Storage Service (S3), Iron Mountain, EMC Atmos Online, Google Cloud Storage et SQL Azure de Microsoft.*
- **Logiciel en tant que service (SaaS)** : Ce service offre un environnement de logiciel complet à distance aux clients, par exemple, des applications de messagerie électronique, de traitement de texte, de gestion des relations clients et bien d'autres types d'applications. Exemples: *Google Docs,*

Calendar et Gmail, Zimbra, Spotify, Salesforce.com, Microsoft Office 365 et SAP Business by Design.

- **Plateforme en tant que service (PaaS)** : Ce service permet aux développeurs de logiciels de créer des applications sur mesure dans les nuages, en tirant profit de la capacité des nuages à fournir automatiquement des ressources informatiques et de stockage supplémentaires lorsque cela est nécessaire. Exemples : *IBM Websphere, Force.com, Springsource, Morphlabs, Google App Engine, Microsoft Windows Azure et Amazon Elastic Beanstalk.*
- **Infrastructure en tant que service (IaaS)** : Ce service permet aux développeurs de logiciels de contrôler directement les ressources informatiques et de stockages fournis par un nuage. Ce service offre davantage de flexibilité au prix d'une plus grande complexité pour tirer profit de l'ensemble des services basés sur le nuage. Exemples : *Elastic Compute Cloud d'Amazon, Zimory, Elastichosts et vCloud Express de VMWare.*

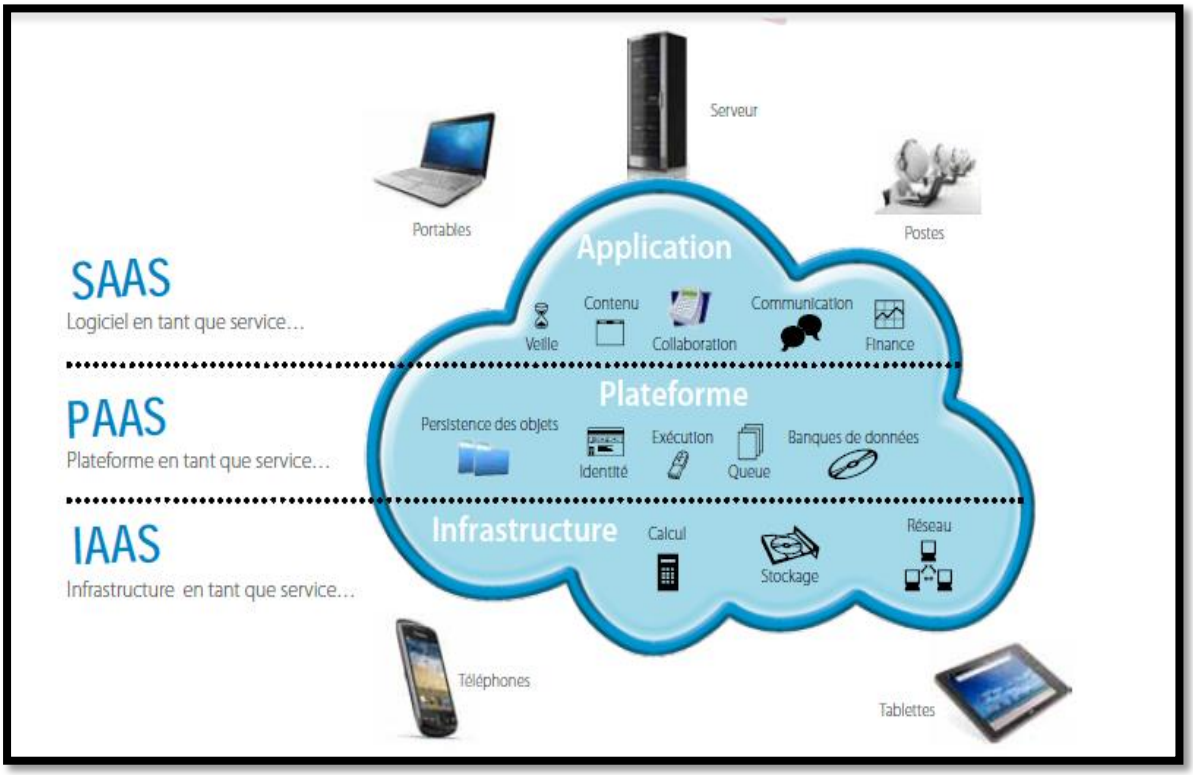


Figure I-11 : **Les différents types de services dans les clouds**

2-4 InterCloud (La fédération de Nuage)

Un nuage correspond a une infrastructure et a son domaine d'administration, de façon plus simple, il est courant d'associer un nuage a l'entreprise qui est responsable a la gestion de l'infrastructure associées. On parlera alors de nuage d'Amazon, et celui de Google …. ; Cependant, du point de vue d'un utilisateur cet ensemble de nuage accessible publiquement via l'internet peut être vue comme un méta-nuage, au sein du quel un certain nombre de services et de ressources informatiques sont disponible. De la même façon qu'Internet et le réseau des réseaux ce méta-nuage et le nuage des nuages et on l'appel **«InterCloud » [11].**

2-4-1 Définition

InterCloud est un ensemble de services et de ressources informatiques qui sont publiquement disponible via Internet ; On retrouve trois grandes catégories de services : le logiciel, la plateforme et l'infrastructure en tant que service.

Le modèle de déploiement des applications distribuées sur les une fédération de nuages correspond au nuage hybride, il a pour particularité d'utilisé plusieurs nuages au sein d'un même environnement de dépoilement.

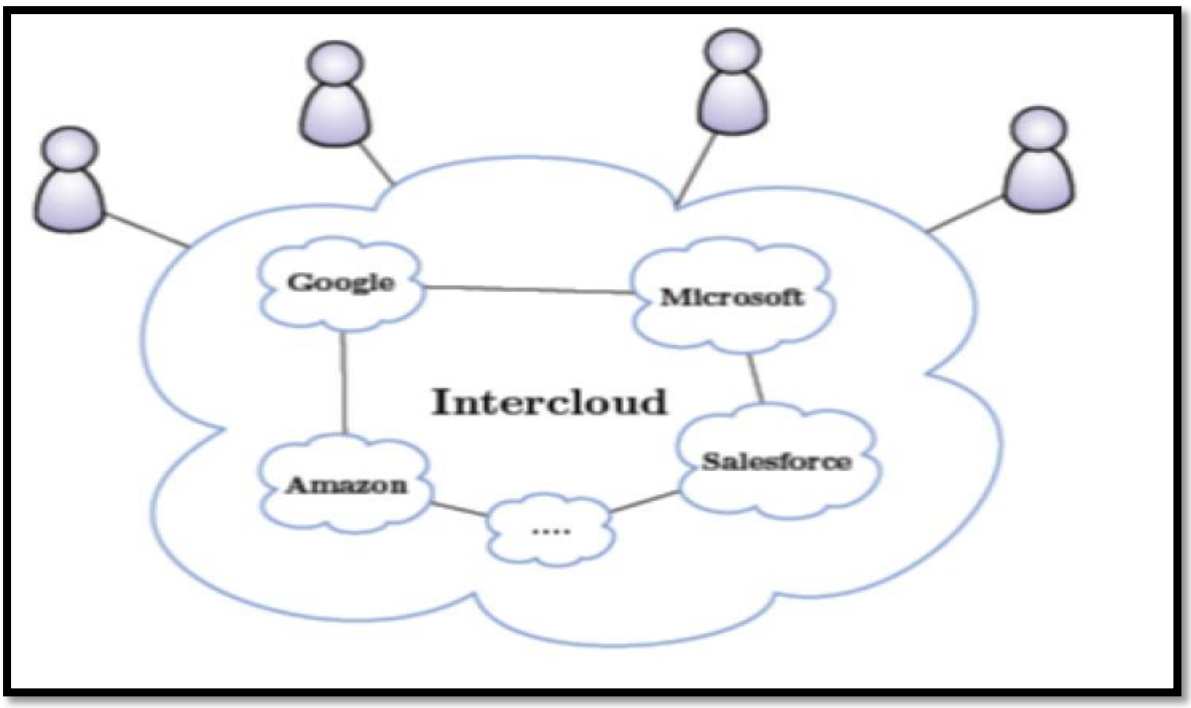


Figure I-12 : **Inter cloud (Le nuage des nuages)**

3) Tableau Comparatif

	Les Grilles	**Les Clusters**	**Les Clouds**
Technologie	✓ Grappes ✓ Batchs	✓ Ordinateurs (les Nœuds)	✓ Datacenters ✓ Virtualisation
La localisation Géographique	✓ Les Réseaux **WAN**	✓ Les Réseaux **LAN**	✓ Les Réseaux **WAN**
Les Ressources	✓ Hétérogènes	✓ Homogènes	✓ Hétérogènes
Gestion de Ressources	✓ Statique	✓ Statique	✓ Dynamique
Modèle de Programmation	✓ MPI (**M**essage **P**assing **I**nterface) ✓ GridRPC (Grid **R**emote **P**rodedure **C**all)	✓ SMPD (**S**ingle **P**rogram **M**ultiple **D**ata)	✓ MapReduce ✓ Scripts ✓ Workflows
Modèle de Sécurité	✓ Certificats	✓ Certificats	✓ Web + SSL*(***S**ecure ***S**ockets ***L**ayer*)*
Des Domaines Applicatifs Différents	✓ HPC(**H**igh **P**erformance **C**omputing) ✓ HTC(**H**igh **T**ech **C**omputer)	✓ HPC (**H**igh **P**erformance **C**omputing)	✓ Business ✓ Web Servers

Conclusion

Les systèmes distribués est apparu suite à la demande des scientifiques de disposer d'une très grande puissance de calcul et d'une capacité de stockage, De même, les scientifiques ont besoin d'effectuer des calculs importants pour modéliser une situation, Ces traitements exigent la plupart du temps des moyens de calculs et de stockage considérables.

Nous avons présenté dans ce chapitre Trois grands systèmes du calcul distribués :

- **Les Grilles Informatiques (Grid Computing) :** Une Grille Informatique destinée à mettre à la disposition des utilisateurs des ressources pour réaliser du calcul distribué et pour stocker des données. Plus concrètement, elle est constituée d'un grand nombre de machines hétérogènes.
- **Les Fermes de Calcul (Clusters) :** On parle de grappe de serveurs ou de ferme de calcul pour désigner des techniques consistant à regrouper plusieurs ordinateurs indépendants appelés Nœuds, afin de permettre une gestion globale et de dépasser les limitations d'un ordinateur pour :

  - Faciliter la gestion des ressources (processeur, mémoire vive, disques dur, bande passante réseau) ;
  - Faciliter la montée en charge ;
  - Permettre une répartition de la charge ;

- **L'Informatique en Nuage (Cloud Computing) :** Le Cloud Computing est un modèle Informatique qui permet un accès facile et à la demande par le réseau à un ensemble partagé de ressources informatiques configurables (serveurs, stockage, applications et services) qui peuvent être rapidement provisionnées et libérées par un minimum d'efforts de gestion ou d'interaction avec le fournisseur du service.

Introduction

L'équilibrage de charge est un élément important lors de la mise en place de services amenés à croître. Il faut s'assurer que la capacité à monter en charge soit la plus optimale possible afin d'éviter toute dégradation que ce soit en terme de performances ou de fiabilité lors d'affluences importantes.

Le principe de base de l'équilibrage de charge (**Load Balancing**) consiste à effectuer une distribution des tâches à des machines de façon intelligente.

Les objectifs du L'équilibrage de charge sont comme suit :

- Amélioration des temps de réponse des services.
- Capacité à pallier la défaillance d'une ou de plusieurs machines,
- Ajout de nouveaux serveurs sans interruption de service **[12]**.

1) Problème de l'équilibrage de charge

Le problème de l'équilibrage étant un problème relativement ancien, beaucoup d'approches ont été proposées pour le résoudre. *Casavant et Kuhl* ont défini une taxonomie largement adoptée par la communauté scientifique dont les principales classes sont **[13] :**

a) *Approche Statique Vs. Approche Dynamique*

Dans une approche statique, les tâches sont assignées aux machines avant l'exécution de l'application qui les contient. Les informations concernant le temps d'exécution des tâches et les caractéristiques dynamiques des machines sont supposées connues a priori. Cette approche est efficace et simple à mettre en œuvre lorsque la charge de travail est au préalable suffisamment bien caractérisée.

Dans une approche dynamique, l'assignation des tâches aux machines se décide durant la phase d'exécution, en fonction des informations qui sont collectées sur l'état de charge du système. Ceci permet d'améliorer les performances d'exécution des tâches mais au prix d'une complexité dans la mise en œuvre de cette stratégie, notamment en ce qui concerne la définition de l'état de charge du système, qui doit se faire de manière continue.

b) *Approche Centralisée Vs. Approche Distribuée*

Dans une approche centralisée, un site du système est choisi comme coordinateur. Il reçoit les informations de charge de tous les autres sites qu'il assemble pour obtenir l'état de charge global du système.

Dans le cas d'une approche distribuée, chaque site du système est responsable de collecter les informations de charge sur les autres sites et de les rassembler pour obtenir l'état global du système. Les décisions de placement de tâches sont prises localement, étant donné que tous les sites ont la même perception de la charge globale du système.

c) *Approche Source-Initiative Vs. Receveur-Initiative*

L'approche source- initiative est appliquée lorsqu'un site, appelé source, détecte qu'il a une surcharge de travail et qu'il cherche à transférer le surplus vers un site faiblement chargé. L'approche receveur initiative s'applique lorsqu'un site faiblement chargé, appelé receveur, demande à recevoir tout ou partie du surplus des sites surchargés.

2) Le système d'équilibrage de charge

Un système d'équilibrage de charge est composé de deux éléments essentiels *politiques* et *mécanismes* **[14].** Les politiques considèrent l'ensemble des choix à effectuer pour distribuer une charge de travail alors que les mécanismes réalisent physiquement la répartition de la charge et fournissent les informations exigées par les politiques.

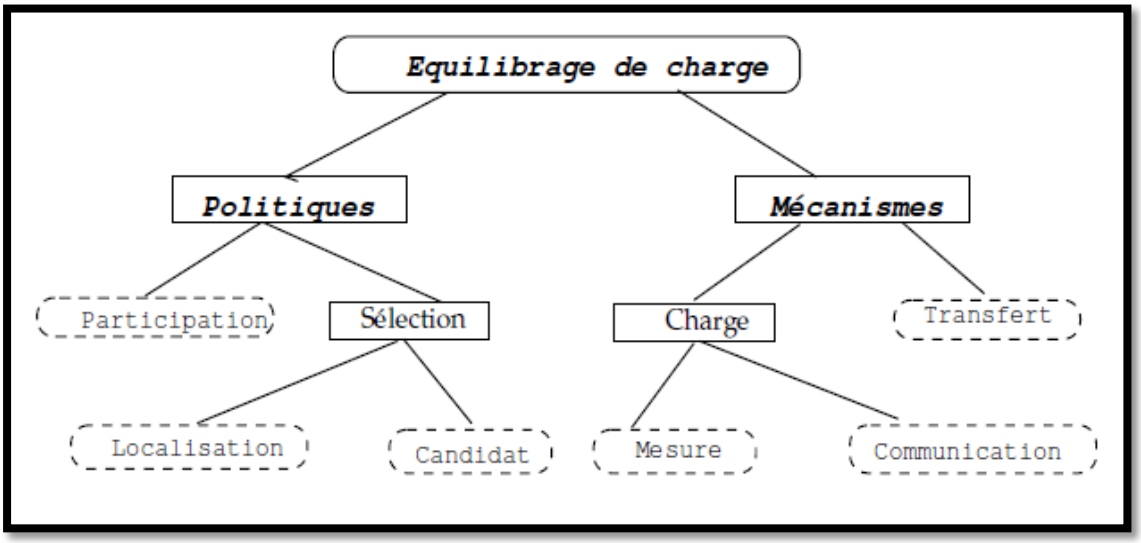


Figure II-1 : **Composants d'un système d'équilibrage de charge**

2-1 Les Politiques

- ***Politique de participation*** **:** Le but de cette politique consiste à déterminer si un site est dans un état approprié pour participer à un transfert de tâches comme *source* (site surchargé) ou comme *receveur* (site sous-chargé).
- ***Politique de sélection de la localisation*** **:** Cette politique est responsable de trouver, pour un site donné, un partenaire (source ou receveur), une fois que la politique de participation a décidé que ce site était soit source, soit receveur.
- ***Politique de sélection des tâches à transférer*** **:** Une fois que les politiques de participation et de localisation ont décidé qu'un site $\mathcal{S}i$ est source et qu'un autre site $\mathcal{S}j$ est receveur, cette politique est responsable du choix des tâches à transférer de $\mathcal{S}i$ vers $\mathcal{S}j$.

2-2 Les Mécanismes

- ***Mécanisme de mesure de la charge*** **:** Dans toute approche d'équilibrage de charge, une des difficultés majeures est celle qui consiste à évaluer la mesure de la charge d'un site. Dans la plupart des travaux existants, c'est la longueur de la file d'attente qui détermine la charge d'un site. Certains auteurs **[15-16]** préconisent comme indicateur de charge, une combinaison entre la longueur de la file d'attente CPU, celle des Entrées/Sorties et l'occupation mémoire. Dans le cas des grilles de calcul, il est nécessaire de tenir compte aussi de l'hétérogénéité des ressources et des réseaux de communication pour mesurer la charge d'un site.
- ***Mécanisme de définition de la charge*** **:** Ce mécanisme essaie de définir la charge globale d'un système en collectant les informations de charge sur l'ensemble ou une partie des sites du système. Il faudra alors définir les méthodes selon lesquelles l'information de charge est collectée puis diffusée aux sites.

4) Les algorithmes d'équilibrage de charge

3-1 Algorithme intra / extra / inter Grappe

A. Stratégie d'équilibrage de charge :

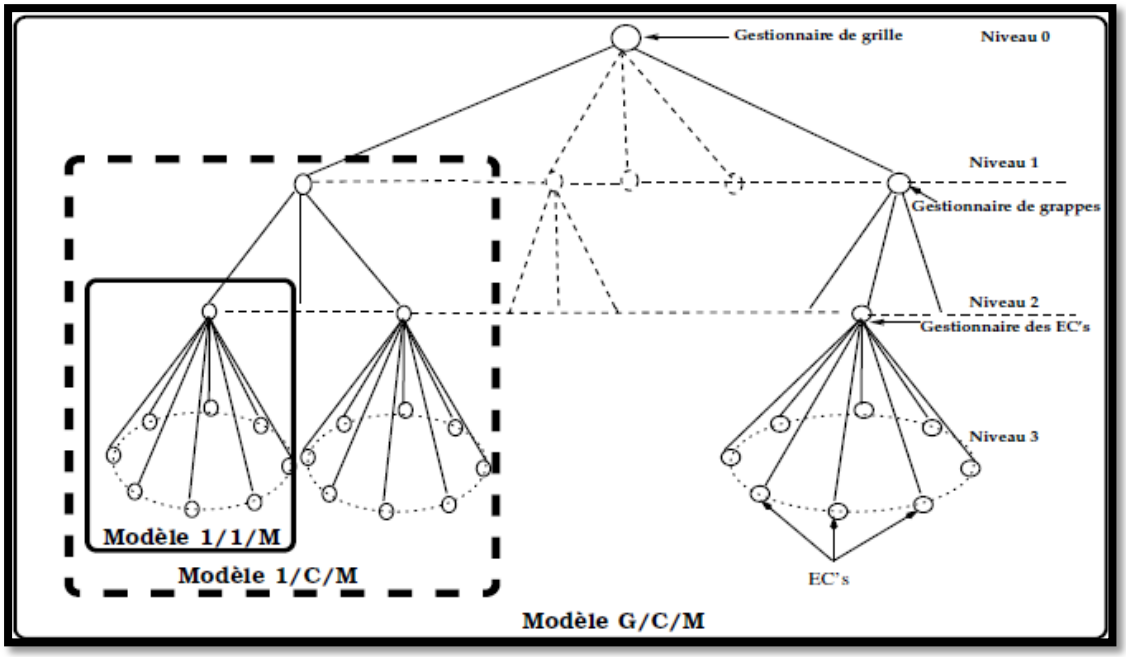


Figure II- 2 : *Modèle générique de représentation d'une grille*

La structure arborescente du modèle proposé par cette approche, permet de développer une stratégie hiérarchique à trois niveaux d'équilibrage : *Intra-Grappe*, *Extra-Grappe* et *Inter-Grappes.*

- ***Équilibrage Intra-Grappe*** : Dans ce premier niveau, chaque gestionnaire d'éléments de calcul décide de déclencher une opération d'équilibrage en fonction de la charge courante de la grappe qu'il gère. Cette charge est estimée à partir des différentes informations de charge envoyées périodiquement par les EC's (éléments de calcul) qui composent la grappe. Le gestionnaire tente, en priorité, d'équilibrer la charge de la grappe localement en la répartissant entre les EC's qui lui appartiennent. Cette approche de localité a pour objectif de réduire les coûts de communication, en évitant les communications extra/inter Grappes.
- ***Équilibrage Extra-grappe*** : Dans ce deuxième niveau, l'équilibrage se fait à l'échelle des extra-grappes. Il intervient dans le cas où certains

gestionnaires des EC's n'ont pas réussi à équilibrer localement leurs charges. Il y aura ainsi transfert de tâches entre grappes surchargées et grappes sous-chargées de la même extra-grappe. Dans un souci de réduire au maximum les coûts de communication, les grappes réceptrices seront sélectionnées en fonction des débits des réseaux.

- ***Équilibrage Inter-grappes*** : Dans ce troisième niveau, l'équilibrage de charge n'est déclenché que si un ou plusieurs gestionnaires de grappes n'arrivent pas équilibrer leurs charges localement entre les grappes qu'ils gèrent. Dans ce cas extrême, il sera alors nécessaire au gestionnaire de grille de transférer un certain nombre de tâches à partir d'extra-grappes surchargées vers d'autres qui sont sous-chargée

B. Principe :

La stratégie d'équilibrage de charge de cet algorithme est dédiée à la classe des tâches indépendantes et composée des trois étapes suivantes **:**

Étape 1 : Estimation de la charge courante du cluster (Resp. de la Grille)

Sachant que N représente le nombre d'éléments de calcul du cluster (resp. le nombre de cluster de la grille), Chaque gestionnaire du cluster exécute les étapes suivantes :

- Estime la charge courante (*LOD*) du cluster (resp. de la grille) en se basant sur les informations de charge envoyées par ses éléments de calcul (resp. par les gestionnaires de clusters de la grille).

$$\mathbf{LOD} = \sum_{I=1}^{N_k} \boldsymbol{LOD\ i}$$

- Estime la capacité (*SAT*) et la vitesse *(SPD*) du cluster (resp. de la grille).

$$\mathbf{SPD} = \sum_{I=1}^{N_k} \boldsymbol{SPD\ i} \qquad \mathbf{SAT} = \sum_{I=1}^{N_k} \boldsymbol{SAT\ i}$$

- Estime le temps d'exécution moyen du cluster (resp. de la grille).

$$\boldsymbol{TEX} = \frac{\boldsymbol{LOD}}{\boldsymbol{SPD}}$$

- Calcul l'écart type (σ) sur les charges de travail de ses éléments de calcul (resp. de l'ensemble des clusters de la grille) dans le but de mesurer l'étendue

des variations de charge entre le cluster et ses nœuds (resp. entre les différents clusters).

$$\sigma = \frac{1}{N}\sqrt{\sum (TEXi - TEX)2}$$

- Envoie son information de charge de travail à l'ensemble des clusters de la Grille.

Etape 2 : prise de décision

Dans cette étape, le gestionnaire décide s'il est nécessaire d'effectuer une opération d'équilibrage de charge ou non. Pour cela, il exécute les deux actions suivantes :

✓ ***Définition de l'état de déséquilibre/saturation/surcharge du cluster***

Soit l'écart type qui mesure la variation moyenne entre le temps d'exécution des éléments de calcul et celui de leur gestionnaire associé, c'est-à-dire un cluster est en état d'équilibre lorsque cet écart est relativement faible. En effet, cela implique que les temps d'exécution de chaque élément de calcul convergent vers le temps d'exécution de son cluster.

- **État d'équilibre :** En pratique, il s'agit de définir un seuil d'équilibre, noté ε, à partir du quel l'écart type tend vers zéro et donc le cluster est en état d'équilibre :

Si ($\sigma \leq \varepsilon$) **Alors** le cluster est équilibré S**inon** il est déséquilibré.

- **État de saturation** : Un cluster peut être déséquilibré tout en étant saturé. Dans ce cas particulier, il n'est pas utile d'entamer un équilibrage local, puisque le cluster restera surchargé. Pour mesurer la saturation d'un cluster, un autre seuil, noté δ, appelé seuil de saturation. Lorsque la charge courante d'un cluster avoisine sa capacité maximale, il est évident qu'il ne sert à rien de l'équilibrer puisque tous ses éléments de calcul sont saturés.

- **État de surcharge :** Si le cluster est en état de saturation, il déterminera son état par rapport aux clusters de la grille. Soit ***TEXG*** le temps d'exécution de la

grille ***TEXi*** le temps d'exécution du cluster. Soit un intervalle de confiance basé sur l'écart type : *[TEXG - σ ; TEXG +σ]*. alors :

Si (TEXi > (TEXG +σ)) **Alors** le cluster est surchargé.

✓ ***Partitionnement***

Pour un cas de déséquilibre, les éléments de calcul surchargés (Sources) ainsi que les éléments de calcul sous-chargés (receveurs), sont déterminés Selon l'écart entre le temps d'exécution de chaque élément de calcul et celui de son cluster ***TEXc***. Ainsi, tout élément est considéré comme :

- Sous-chargé si son temps d'exécution est inférieur à ***TEXc - σ;***
- Surchargé si son temps d'exécution est supérieur à ***TEXc + σ;***

Pour un cas de surcharge du cluster, les clusters sous chargés, selon l'écart entre le temps d'exécution de chaque cluster et celui de la grille estimer par le cluster surchargé. Ainsi, Un cluster est sous-chargé si son temps d'exécution est inférieur à ***TEXG - σ.***

Etape 3 : Transfert de tâches

Afin de transférer des tâches à partir des éléments de calculs (resp. clusters) surchargés vers les éléments de calculs (resp. clusters) sous-chargés, nous proposons l'heuristique suivant :

- Calculer la disponibilité, en termes de capacité de calcul, qui correspond à la charge totale offerte par les éléments de calcul (resp. clusters) receveurs.

$$\textbf{Offre} = \sum_{Er \epsilon GER} \frac{LOD\ k\ .\ SPD\ ik}{SPD\ k} - LOD\ ik$$

- Calculer la demande, c'est-à-dire, la charge totale requise par l'ensemble des éléments de calcul (resp. cluster) sources.

$$\textbf{Demande} = \sum_{Es \epsilon GES} LOD\ ik - \frac{LOD\ ik\ .\ SPD\ ik}{SPD\ ik}$$

- Si l'offre est très inférieure à la demande (l'offre n'est pas en mesure de satisfaire suffisamment la demande), il n'est pas recommandé d'entamer un équilibrage local (resp équilibrage global).

Nous introduisons un troisième seuil, appelé seuil d'espérance et dénoté ρ, afin de mesurer l'écart relatif entre l'offre et la demande. Nous pouvons écrire l'expression suivante :

Si (Offre /Demande >ρ) **Alors** Effectuez un équilibrage local (resp. global)

Sinon Effectuer un équilibrage global (resp. ne rien faire).

- Effectuer un transfert de charge en tenant compte des coûts de communication dans le cas particulier d'un équilibrage global.

Algorithme d'équilibrage Intra/Extra/Inter-grappe : (cas d'un groupe G)

Estimation de la charge du groupe

1. Collecte périodique de l'information de charge

Pour chaque élément Ei de G**faire**

Envoi de sa charge actuelle $LODi$ à son gestionnaire associé.

Fin Pour

2. Le gestionnaire de groupe Geffectue les opérations suivantes :

a- Estimer la vitesse $SPDG$ et la capacité $SATG$ du groupe G ;

b- Calculer la charge courante $LODG$ et le temps d'exécution $TEXG$ de G ;

c- Calculer l'écart type σ sur les temps d'exécution des éléments de G ;

d- Envoyer l'information de charge de Gà son gestionnaire correspondant.

Prise de décision

3. Test d'équilibre :

a- Cas intra-grappe : **Si** ($\sigma_G \leq \varepsilon$) **Alors** grappe en état d'équilibre **Fin Si**

b- Autres cas : **Si** (proportion d'éléments surchargés · seuil donné) **Alors**

le groupe Gest en état d'équilibre **Fin Si**

4. Test de saturation :

Si ($\frac{LOD\ G}{SPD\ G} > \delta$) **Alors** G est saturé **Fin Si**

5. Partitionnement des éléments de Gen surchargés (GES), sous-chargés (GER) et

équilibrés (*GEN*) : *GES*←φ; *GER*←φ,*GEN*←φ.

Pour Chaque *Ei* de *G* **faire**

Si (*Ei* Saturé) **Alors**

GES←*GES*∪ *iE*

Sinon

Selon que

TEXi > $TEX_G + \sigma_G$: *GES*←*GES*∪ *iE*

TEXi < $TEX_G - \sigma_G$: *GER*←*GES*∪ *iE*

TEXG − $\sigma_G \leq TEXi \leq TEXG + \sigma_G$: *GEN*←*GEN*∪ *iE*

Fin Selon que

Fin Si

Fin Pour

Transfert de tâches

6. $OFFRE = \sum_{Nik \epsilon GER} \frac{LOD\ k\ .\ SPD\ ik}{SPD\ k} - LOD\ ik$

Demande = $\sum_{Nik \epsilon GER} LOD\ ik - \frac{LOD\ ik\ .\ SPD\ ik}{SPD\ ik}$

Si ($\frac{\text{offre}}{\text{demande}} \leq \rho$) **Alors** Échec d'équilibrage local **Fin Si**

7. Transfert de tâches :

Utiliser l'heuristique 1 en cas de transfert Intra-grappe

Autrement utiliser l'heuristique 2.

3-2 Algorithme des Grilles de calcul (classe des tâches dépendantes)

a. Principe

On vise à améliorer la stratégie précédente dans le but de l'adapter à la classe des tâches dépendantes. Pour cela, nous apercevons chaque méta-tâche comme une seule tâche indépendante des autres. La stratégie d'équilibrage de charge proposée est constituée des trois étapes suivantes :

- **Étape 1 : Estimation de la charge courante du Cluster (resp. de la Grille)**

Sachant que N représente le nombre d'éléments de calcul du cluster (resp. le nombre de cluster de la grille) :

1- Chaque élément de calcul (resp. gestionnaire de cluster) sa charge courante en prenant en considération uniquement les **tâches prête à être exécutée** (avec 0 contraintes à satisfaire).

2- Chaque gestionnaire du cluster exécute les étapes suivantes :

- Estime la charge courante (*LOD*) du cluster (resp. de la grille) en se basant sur les informations de charge envoyées par ses éléments de calcul (resp. par les gestionnaires de clusters de la grille).

$$\mathbf{LOD} = \sum_{I=1}^{N_k} \boldsymbol{LOD\ i}$$

- Estime la capacité (*SAT*) et la vitesse *(SPD*) du cluster (resp. de la grille).

$$\mathbf{SPD} = \sum_{I=1}^{N_k} \boldsymbol{SPD\ i} \qquad \mathbf{SAT} = \sum_{I=1}^{N_k} \boldsymbol{SAT\ i}$$

- Estime le temps d'exécution moyen du cluster (resp. de la grille).

$$\boldsymbol{TEX} = \frac{\boldsymbol{LOD}}{\boldsymbol{SPD}} + \sum_{i=1}^{p} \boldsymbol{Tcom\ i}$$

Avec :

Tcom : temps de communication des informations transmises par les tâches prête à être exécutées et celle qui dépendent d'elles.

P : Nombre de tâches prête à être exécutées.

- Calcul l'écart type (**σ**) sur les charges de travail de ses éléments de calcul (resp. de l'ensemble des clusters de la grille) dans le but de mesurer l'étendue des variations de charge entre le cluster et ses noeuds (resp. entre les différents clusters).

$$\sigma = \frac{1}{N} \sqrt{\sum (\boldsymbol{TEXi} - \boldsymbol{TEX})2}$$

- Envoie son information de charge de travail à l'ensemble des clusters de la grille.

- **Etape 2 : prise de décision**

Dans cette étape, le gestionnaire décide s'il est nécessaire d'effectuer une opération d'équilibrage de charge ou non.

Pour cela, il exécute les deux actions suivantes :

- *Définition de l'état de déséquilibre/saturation/surcharge du cluster*

Si nous considérons que l'écart type mesure la variation moyenne entre le temps d'exécution des éléments de calcul et celui de leur gestionnaire associé, nous pouvons dire qu'un cluster est en état d'équilibre lorsque cet écart est relativement faible. En effet, cela implique que les temps d'exécution de chaque élément de calcul convergent vers le temps d'exécution de son cluster. Ensuite, nous définissons les états d'équilibre et da saturation.

État d'équilibre : En pratique, il s'agit de définir un seuil d'équilibre, noté ε, à partir duquel nous pouvons dire que l'écart type tend vers zéro et donc le cluster est en état d'équilibre. Ainsi nous pouvons écrire :

Si ($\sigma \leq \varepsilon$) **Alors** le cluster est équilibré S**inon** il est déséquilibré.

État de saturation : Un cluster peut être déséquilibré tout en étant saturé. Dans ce cas particulier, il n'est pas utile d'entamer un équilibrage local, puisque le cluster restera surchargé.

Pour mesurer la saturation d'un cluster, nous définissons un autre seuil, noté δ, que nous appellerons seuil de saturation.

Lorsque la charge courante d'un cluster avoisine sa capacité maximale, il est évident qu'il ne sert à rien de l'équilibrer puisque tous ses éléments de calcul sont saturés.

État de surcharge : Si le cluster est en état de saturation, il déterminera son état par rapport aux clusters de la grille.

Soit ***TEXG*** le temps d'exécution de la grille ***TEXi*** le temps d'exécution du cluster. Nous définissons un intervalle de confiance basé sur l'écart type: *[TEXG - σ; TEXG +σ]*. Ainsi nous pouvons écrire :

Si (TEXi > (TEXG +σ)) **Alors** le cluster est surchargé.

- *Partitionnement*

Pour un cas de déséquilibre, nous déterminons les éléments de calcul surchargés (sources) ainsi que les éléments de calcul sous-chargés (receveurs), Selon l'écart entre le temps d'exécution de chaque élément de calcul et celui de son cluster ***TEXc***. Ainsi, tout élément est considéré comme :

Sous-chargé si son temps d'exécution est inférieur à ***TEXc - σ;***

Surchargé si son temps d'exécution est supérieur à ***TEXc*** **+ σ;**

- Pour un cas de surcharge du cluster, nous déterminons les clusters sous chargés, selon l'écart entre le temps d'exécution de chaque cluster et celui de la grille estimer par le cluster surchargé. Ainsi, Un cluster est sous-chargé si son temps d'exécution est inférieur à ***TEXG*** **- σ.**

- **Etape 3 : Transfert de tâches**

Afin de transférer des tâches à partir des éléments de calculs (resp. clusters) surchargés vers les éléments de calculs (resp.clusters) sous-chargés, nous proposons l'heuristique suivant :

a- Calculer la disponibilité, en termes de capacité de calcul, qui correspond à la charge totale offerte par les éléments de calcul (resp. clusters) receveurs.

$$\textbf{Offre} = \sum_{Nik \epsilon GER} \frac{LOD\ k\ .\ SPD\ ik}{SPD\ k} - LOD\ ik$$

b- Calculer la demande c'est a dire la charge totale requise par l'ensemble des éléments de calcul (resp. cluster) sources.

$$\textbf{Demande} = \sum_{Nik \epsilon GES} LOD\ ik - \frac{LOD\ k\ .\ SPD\ ik}{SPD\ k}$$

c- Si l'offre est très inférieure à la demande (l'offre n'est pas en mesure de satisfaire suffisamment la demande), il n'est pas recommandé d'entamer un équilibrage local (resp équilibrage global).Nous introduisons un troisième seuil, appelé seuil d'espérance et dénoté ρ, afin de mesurer l'écart relatif entre l'offre et la demande. Nous pouvons écrire l'expression suivante :

Si (Offre /Demande >ρ) **Alors** Effectuez un équilibrage local (resp. global)

Sinon Effectuer un équilibrage global (resp. ne rien faire).

d- Effectuer un transfert de charge en tenant compte des coûts de communication dans le cas particulier d'un équilibrage global. Le transfert ce fait par **méta-tâches**, pour ne pas accroître le temps de communication *Tcom*.

3-3 Algorithme de Diffusion

A. Principe de fonctionnement de l'algorithme de diffusion génétique

Les algorithmes de diffusion diffusent la charge initiale de proche en proche sur les autres nœuds jusqu'à l'obtention d'une charge équitable sur l'ensemble des nœuds.

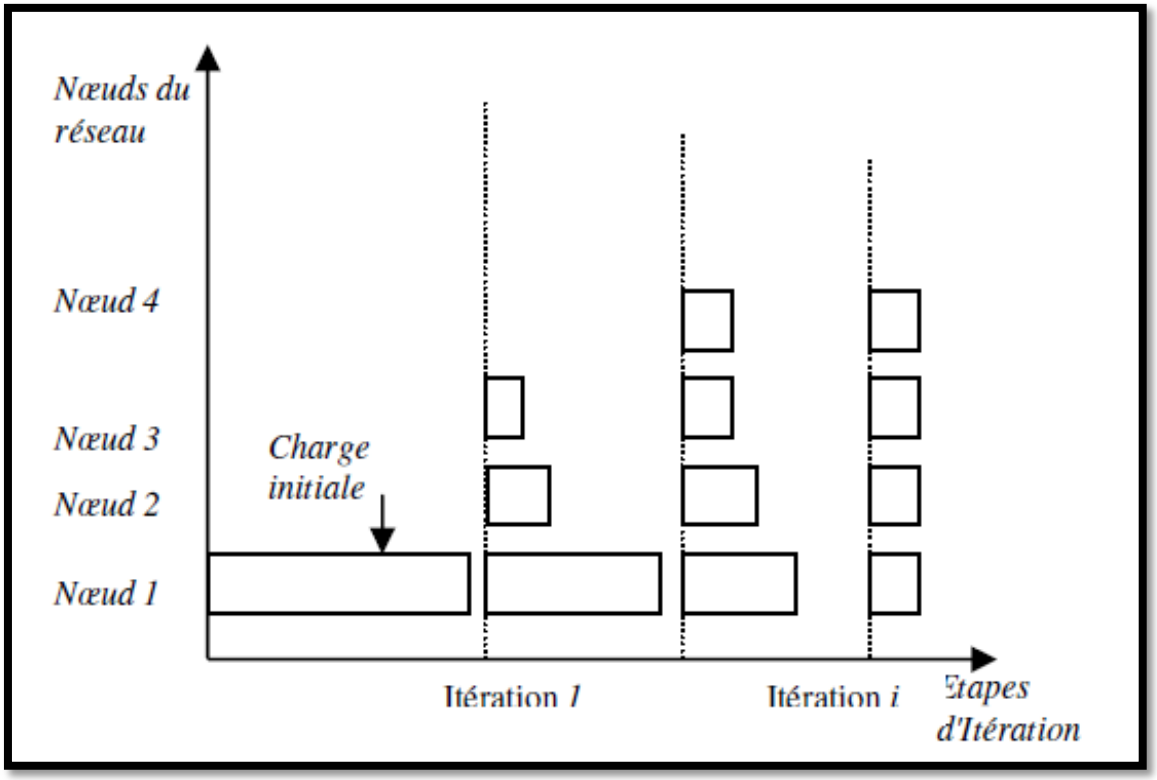


Figure II-3 : **Principe des algorithmes de diffusion**

Dans l'algorithme de diffusion génétique, chaque nœud partage sa charge en deux portions ; il garde une portion pour lui et il diffuse l'autre portion à ses voisins d'une façon équitable, selon le principe de la méiose (méiose ; division cellulaire qui, à partir d'une cellule somatique, produit deux cellules filles, toutes deux identiques, et également identiques à la cellule mère) *(La figure. II-4).*

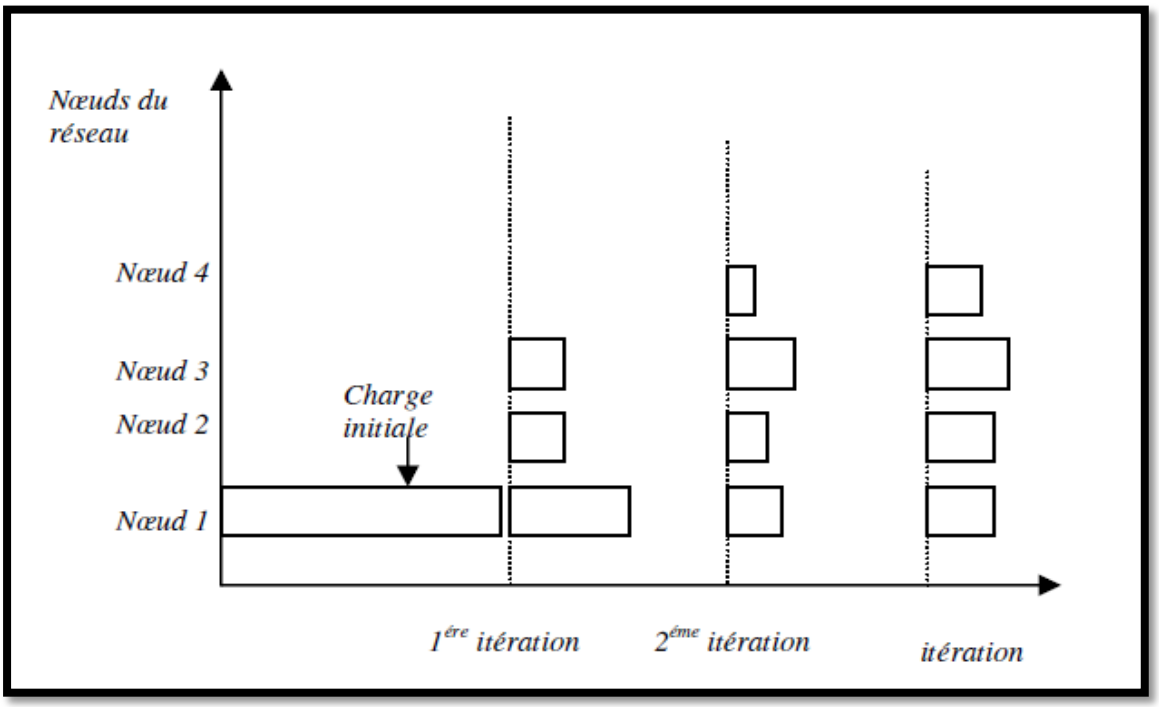


Figure II-4 : **Principe d'algorithme de diffusion génétique**

B. Modélisation de l'algorithme de diffusion génétique

L'algorithme de diffusion trouve une formulation mathématique comme suit :

$$Wi^{(t+1)} = wi^{(t)} + \sum \alpha ij\,(wj^{(t)} - wi^{(t)}) + \mu_i^{(t+1)} - c \qquad (1)$$

avec :

$wi^{(t)}$:la charge du processeur *i* à l'instant *t*.

αij :est un paramètre d'échange dans le cas d'algorithmes d'équilibrage de charge de type diffusion ; il représente une fraction de la différence de charge entre les processeurs *i* et *j*.

$\mu_i^{(t+1)}$ *:* est la charge de travail qui est créée sur le processeur *i* à l'instant *t*.

c : est une constante représentant la charge de travail consommée par un processeur en une itération.

L'équation (1) peut être simplifiée, si l'on considère que la charge est statique, c'est à dire la charge demeure constante entre chaque itération.

$$\left(\sum_{i=1}^{n} wi(t) = \sum_{i=1}^{n} wi(t+1)\right)$$

Pour une charge statique, le modèle s'exprime simplement en considérant que $\mu_i^{(t+1)}=0$ à tout instant *t* et *c* = 0. Dans ce cas, l'équation (1) peut se mettre sous la forme: $W(t+1) = MW(t)$

Où : *W*(*t*) est le vecteur de dimension *n* contenant la charge de tous les processeurs à l'instant *t.*

M est la matrice de diffusion définie par *mij* telle que :

$$M_{ij}=\begin{cases} \alpha ij \; si \; i \neq j \\ 1-\sum \alpha ij \quad si \quad i=j \; (i,j=1..n) \end{cases}$$

C. Détermination de la matrice de diffusion génétique

La matrice de diffusion génétique que nous avons définie s'écrit comme suit:

$$M_{ij}=\begin{cases} \alpha ij=\frac{1}{2Vt} \; si \; i \neq j \\ \frac{1}{2} \; si \; i=j \\ 0 \; si \; l'arc \; (i,j) \epsilon dans \; E \end{cases}$$

Dans la matrice de diffusion génétique, la somme de αij sur les colonnes donne *1*, ce qui assure une probabilité sur les colonnes $\sum_{i=1}^{n} \alpha ij = 1$, contrairement aux algorithmes classiques de diffusion où la somme de probabilité est égale à *1* sur les lignes.

La matrice de diffusion proposée permet de générer une population initiale d'individus où chacun représente une solution au problème, à savoir une allocation possible. Dans notre cas on prend la charge initiale comme une population initiale et on reproduit cette charge à chaque itération grâce aux éléments *Mij*.

D. Etapes principales de l'algorithme proposé :

Les algorithmes de diffusion supposent qu'un nœud d'un graphe peut envoyer et recevoir une charge de/vers tous ses voisins simultanément à chaque itération. Après un certain nombre d'itérations un état équilibré est atteint. Ces algorithmes itératifs sont souvent utilisés en raison de leur caractère totalement distribué et de leur généricité. Ce qui leur permet d'être appliqués sur n'importe quel système nécessitant un équilibrage de charge**.**

Lorsque un déséquilibre survient ou le seuil maximum/minimum est atteint sur un nœud du système, l'algorithme d'équilibrage doit répartir la charge, donc les données, de telle sorte que le temps global d'exécution soit réduit.

On note tout de même que ces algorithmes exigent quelques contraintes d'entrées :

- Nombre de nœuds : l'information de l'ensemble des processeurs du réseau est nécessaire pour répartir la charge du travail.
- Les seuils min et max : la charge très faible ou la charge très haute permet le déclenchement du processus d'équilibrage ainsi que l'arrêt de ce dernier.
- Etat du système à équilibrer : la répartition de la charge initiale du système à équilibrer egst fondamentale pour pouvoir commencer le processus d'équilibrage.

Le diagramme suivant donne les principales phases de l'algorithme proposé:

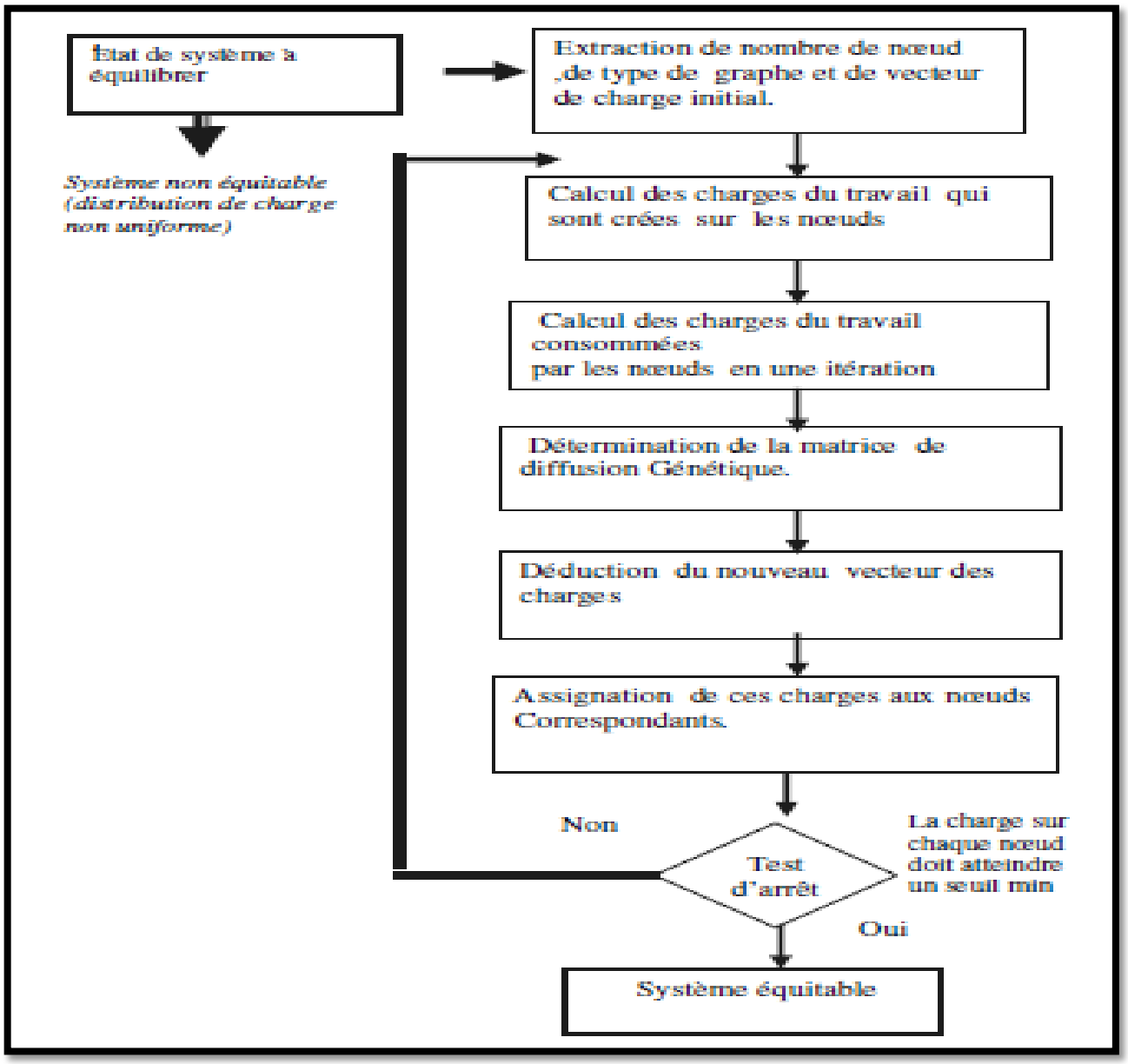


Figure II-5 : **Les étapes principales de l'algorithme**

3-4 Algorithme d'équilibrage de charge (Cloud Computing) [17]

- ***Objectifs d'optimisation***

Dans l'informatique en nuage (Cloud Computing), la planification des tâches est un multi-objectif de problème d'optimisation. L'objectif d'optimisation peut être aussi suivre :

- ✓ Réduire l'emploi enjambant. Du point de vue abonnés, l'emploi du temps d'achèvement devrait pas au-delà de leur la tolérance. Donc, nous devrions minimiser la fin totale de l'emploi du temps. En outre, moyenne durée d'emploi de marque ne peut être négligé.
- ✓ Charge équilibré. Pour autant que l'opérateur est concerné, les ressource est distribué efficacement et équitablement et enfin l'amélioration de l'utilité des ressources.

- ➢ ***Équipe modèle- Allocation***

En environnement de l'informatique en nuage, l'architecture de la répartition des tâches est un modèle à trois niveaux

- ✓ demande de tâche : les utilisateurs interagissent avec le système dans la couche intermédiaire
- ✓ la gestion des ressources : la couche intermédiaire, la gestion des ressources joue un remarquable rôle dans l'emploi de division en tâches, tâches et répartition la gestion des ressources
- ✓ l'exécution de la tâche : dans la dernière couche les tâches sont exécutées en tant que des séquences prédéfinies.

La gestion des ressources est la clé allocation des tâches. Afin d'analyser le temps total d'exécution des travaux soumis, nous quantifions d'abord le nombre de tâches

Emploi **t** comme **TaskNum (t)**, qui est donné par la formule **(1)**,

où **J** est le nombre d'emplois totaux,

N représente le n ° de nœuds de travailleurs.

$$\text{TaskNum (t)} = \sum_{t=1}^{j} Tasknum(t) \qquad \textbf{(1)}$$

Pour distinguer ces tâches, nous identifions la tâche **j ème** dans **i ème** Emploi avec m, donnée par la formule (2)

$$m = \sum_{k=1}^{i-1} Tasknum(K) + j \qquad \textbf{(2)}$$

Comme la capacité de calcul d'un nœud unique est certain, on peut estimer le temps d'exécution possible une fois les tâches sont mettre sur eux. Ici, nous introduisons la matrice ETC (tasknum × workernum) (Durée prévue à calculer) pour décrire le temps consommer de la tâche **i ème** exécuté sur le travailleur **j éme**, où **tasknum** se réfère à le Nombre des tâches, et workernum exprime le nombre de travailleurs. Ainsi, selon décodée la relation entre les travailleurs et les tâches et la matrice ETC, nous travaillons sur le temps d'exécution de la tâche de **t éme**

$$\text{Job Time (t)} = \max \sum_{j=1}^{k} \text{TaskTime (j, i)} \qquad \textbf{(3)}$$

Où k désigne la tâche **i** dans l'emploi **t** la position alloué, **TaskTime (j, i)** désigne le coût d'exécution de la tâche **i** sur le nœud **j**.la durée moyenne d'un travail d'exécution est donnée par la formule (4). nous ne tenons pas compte de l'échec nœud tout traitement.

$$\text{avgJobTime} = \frac{\sum_{t=1}^{j} \text{JobTime (t)}}{j} \qquad \textbf{(4)}$$

Enjambant tâche comprend trois sections, à savoir, la tâche informatique temps, le temps E / S et le temps de transmission de données à distance. Pour une tâche **i, 1 ≤ i ≤ taskNum (t)**, **Ti** désigne le temps de tâche i ème et est donnée par

$$T_i = T_{\text{computing i}} + T_{\text{diskI / O i}} + T_{\text{ransmission i}} \qquad \textbf{(5)}$$

$\mathbf{T_{computing}}$ est relativement courte par rapport aux autres temps-coûts en grappes. Disk I / O temps représente le temps nécessaire pour le transfert de données à partir du disque local, il est certain de particulier environnement matériel. $\mathbf{T_{transmission}}$ représente des données à distance transférer coût du temps. C'est un facteur clé dans la consommatrice de temps total. Considérant localité des données, le système choisit habituellement attentivement le nœud de l'exécution des tâches et la fourniture de données. Mais dans certaines des

cas, il n'est pas évité à exécuter sur le nœud au coût de télétransmission de données. Dans les circonstances, nous pouvons estimer d'environnement réseau actuel. Dans cet article, latence du réseau n' est pas prise en compte, nous utilisons la matrice DTC (taskNum × N) pour afficher le coût de transmission de données, où DTC [i, j] représente le coût de la tâche i ème prévu à la j ème nœud, s'il ya une réplication des données sur le j ème nœud, DTC [i, j] = 0. Dans cet article, Donc, le temps total de consommer des tâches est comme suit:

$$\text{Total Time} = \max \sum_{i=1}^{n} noeud\ (w, i) \qquad \textbf{(6)}$$

Lorsqu'un nœud (w, i) désigne le coût d'un temps de la tâche i ème exécutée sur le nœud de w ème, qui, y compris l'informatique, le disque I / O et la transmission des données à distance. Et n est le nombre total de tâches sur ce nœud.

- *Expression de charge*

La charge d'un nœud peut être obtenue par l'addition des charges de VM exécutées sur elle. L'analyse de la séquence décodée, nous pouvons travailler sur le nombre de tâches en cours d'exécution sur le nœud, en fait fonctionnant sur les machines virtuelles. Au cours de cette Total Time durée, nous pouvons travailler à la charge des machines virtuelles. Supposons la charge des machines virtuelles est relativement stable pendant la période, alors nous pouvons définir la charge de VM N ° i dans Total Time est **V (i, Total Time),** et en attendant, il ya K VM fonctionne sur le nœud j. Par conséquent, nous pouvons conclure que, dans TotalTime, la charge du nœud j est

$$\text{PJ (i, Total Time)} = \sum_{i=1}^{k} V(i, totaltime) \qquad (7)$$

Au cours de cette Total Time, la charge moyenne de tous les nœuds est représentée comme suit:

$$\overline{P\ (T)} = \frac{1}{N} \sum_{j=1}^{N} p(j, T) \qquad (8)$$

Pour bien décrire l'intensité de la charge de nœud différent, nous introduire la variance:

$$\alpha\ (\text{T}) = \sqrt{\frac{1}{N-1} \sum_{j=1}^{N} (p(j, T) - \overline{p(T)})^2} \qquad (9)$$

Il est évident que plus le plus l'équilibrage de charge et ordonnancement des tâches raisonnable. L'objectif de cet article est de rechercher le meilleur équilibre de la charge horaire de décision et peu de temps consommé.

- **Les meilleurs algorithmes génétiques**

Les algorithmes génétiques sont une classe d'algorithmes de recherche optimaux qui simuler l'évolution biologique et le mécanisme génétique.

Après la population initiale est produit, il évolue mieux et meilleures solutions approximatives basées sur remise en forme de génération en génération. Au cours de chaque génération, l'individu est choisi basé sur l'aptitude des différents individus dans un certain domaine du problème. Ensuite, les individus combinent, croix et varient par les opérateurs génétiques de la génétique naturelle, puis une nouvelle population représentant un nouvel ensemble de la solution est produite.

Algorithme 1 JLGA

Entrée :
Max : max itérations
S: l'échelle de la population;
N: numéro du nœud de travailleur;
J: nombre d'emplois
C1, C2: Poids de l'emploi total de panoramique et de travail moyen enjambant, C1 + C2 = 1
λ1, λ2: Probabilité de fitness1 et Fitness2,λ1 + λ2 = 1.
Sortie :
Elite $_{1, N}$: La meilleure solution
1: itération, λ ← 0; ← de remise en forme?
2: elite$_{1, N}$, temp$_{1, N}$, P1, P2, P, N, Fitness1, S ←?
3: p ← Initialisation
4: while itération<max do
5: λ = random (0,1)
6: if λ <λ2 then
7: fitness= Fitness2
8: else
9: fitness= Fitness1
10: end if
11: for i = 1 à S do
12: Fitness$_i$ ← Fitness (i)
13: end for
14: elite$_{1, N}$ ← individuelle avec la meilleure condition physique
15: P1, P2 ← 0
16: calculer p_c, p_m
17: if random (0, 1) <p_c then
18: temp$_{1, N}$ ← cossover (p1, p2)
19: end if
20: if random (0, 1) < p_m then
21: P$_{1, N}$ ← mutation (temp)
22: end if
23: itérateur ← itérateur + 1
24: end while

Algorithme 2 Initialisation Greedy (IG)

Entrée:
S: l'échelle de la population;
N: numéro du nœud de Travailleur;
J: Non d'emplois
Sortie:
$P_{s, N}$: population initialisée
1: $P_{s, N} \leftarrow \emptyset$
i, j ← 1
2: while i ≤ S do
3: while j ≤ N do
4: s' il ya un nœud g avec la réplication de données de k nécessaire sur elle et g est inactif alors
5: Site ← g
6: else
7: Choisissez le nœud moindre coût que l'exécution nœud f
8: end if
9: $P_{i, k}$ ← site
10: j ← i + 1
11: end while
12: i ← j + 1
13: end while
14: retour $P_{s, N}$

Conclusion

Ce chapitre propose une stratégie pour résoudre le problème d'équilibrage de charge .Tout d'abord Nous avons présenté les éléments du système d'équilibrage de charge, ensuite nous avons développé l'algorithme (intra, extra, inter) grappe et l'algorithme de diffusion génétique développé, qui optimise bien la vitesse de convergence. La diffusion permet une accélération de la convergence d'équilibrage dès les premières itérations. Enfin les Algorithmes d'équilibrage de charge pour chaque plateforme (Cloud Computing, Cluster, Grid).

Introduction

En termes de modèles de programmation et de mise en œuvre, le parallélisme est une réponse naturelle aux contraintes d'équilibrage de charge dans les systèmes distribués. Nous pouvons noter que la très large majorité des architectures dans ce domaine, sont des architectures parallèles.

En d'autres termes, le parallélisme peut être défini par l'ensemble des modèles de calcul, architectures et techniques permettant de résoudre efficacement un problème calculatoire par la mise en œuvre simultanée de plusieurs unités de traitement **[18].**

Le parallélisme s'oppose aux méthodes dites séquentielles, pour lesquelles la résolution d'un problème est assurée par une séquence d'instructions élémentaires sur un flux de données.

1) Définition du parallélisme

Le principe de base du parallélisme est d'utiliser plusieurs ressources (Processeurs ou Mémoires) qui fonctionnent concurremment pour accroitre la puissance du calcul pour la résolution d'un même algorithme pour un problème donné. L'objectif du parallélisme est non seulement résoudre des problèmes le plus rapidement possible, mais aussi de pouvoir résoudre des problèmes de grandes taille. Les synthèses parallèles de grande taille, comme les Grilles de calcul sont capables d'exécuter rapidement les calculs scientifiques demandant plusieurs mois de calcul sur un seul processeur **[19]**.

2) Classification des Architectures Parallèles

Ci-dessus un tableau qui résume les différentes architectures parallèles **[20]** :

	1 Flux d'instruction	**> 1 Flux d'instruction**
1 Flux d'instruction	*SISD*	*MISD (Pipeline)*
> 1 Flux d'instruction	*SIMD*	*MIMD*

***S :** Single, **M :** Multiple*

***I** : Instruction, **D :** Data*

SISD *:* Une instruction - une donnée machine séquentielle.

SIMD : Plusieurs données traitées en même temps par une seule instruction. Utilisé dans les gros ordinateurs vectoriels.

MISD : Une donnée unique traitée par plusieurs instructions. Architecture pipeline.

MIMD : Exécution d'une instruction différente sur chaque processeur pour des données différentes. Pour simplifier la programmation on exécute la même application sur tous les processeurs.

3) Les types d'architectures parallèles

Il existe deux types de machine parallèle **[20]**

3-1 Les machines à mémoire partagée

Ce type d'architecture est caractérisé par : plusieurs processeurs avec des horloges indépendantes ; et une seule mémoire commune pour tous les processeurs.

Les machines à mémoire partagée permettent de réaliser le parallélisme de données et de contrôle.

Le programmeur n'a pas besoin de spécifier la distribution des données sur chaque processeur, Il définit seulement la partie du programme qui doit être parallélisée et doit gérer les synchronisations.

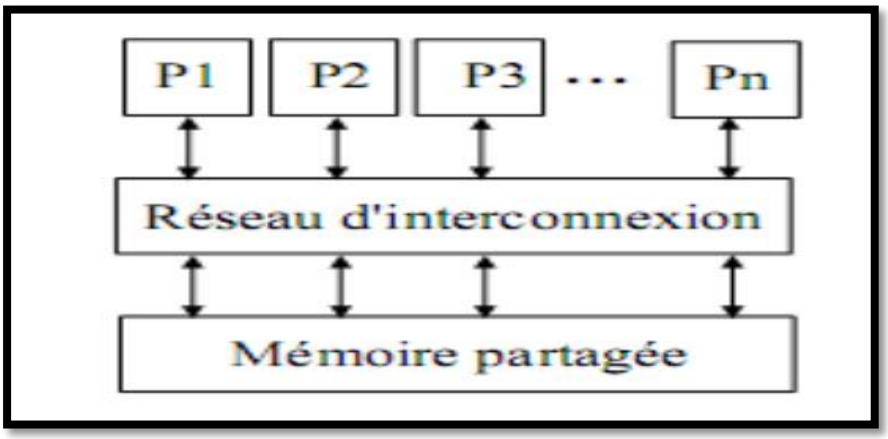


Figure III-1 : **Architecture a mémoire partagée**

3-2 Les machines à mémoire distribuée

Ce type d'architecture est caractérisé par une interconnexion (réseau local rapide) de systèmes indépendants (nœuds), une mémoire propre locale à chaque processeur, aussi chaque processeur exécute des instructions identiques, sur des données identiques, enfin un parallélisme par échange de messages.

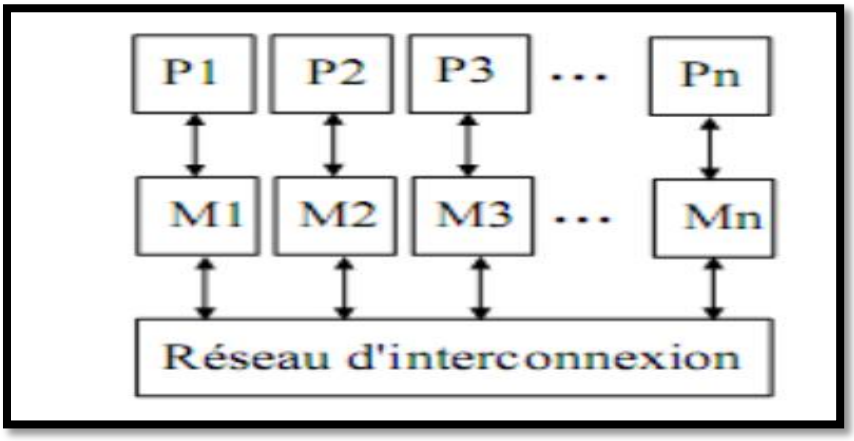


Figure III-2 : **Architecture a mémoire distribuée**

5) La programmation parallèle

Le développement de la programmation parallèle n'est pas un sujet nouveau, mais il était réservé jusqu'au présent à un certain nombre d'experts qui travaillent encore actuellement dans le monde du High Performance Computing (HPC), pour principalement croquer des nombres à partir de supercalculateurs.

4-1 Définition :

La programmation parallèle consiste à développer des programmes qui s'exécutent simultanément sur plusieurs processeurs à la fois.

La programmation parallèle c'est-à-dire que plusieurs processus ou threads sont en mesure de s'exécuter en parallèle avec la possibilité de s'échanger de l'information et de se synchroniser **[21]**.

4-2 Les Modèles du Programmation Parallèle [21]

Un modèle de programmation parallèle est une abstraction du matériel que le programmeur utilise pour coder et exécuter des programmes parallèles. Il implique les langages, les systèmes de communication, les bibliothèques et les compilateurs.

L'intérêt de faire de la programmation parallèle est :

- ✓ Réduire le temps de restitution ;
- ✓ D'effectuer de plus gros calculs ;
- ✓ D'exploiter le parallélisme des processeurs modernes (multi-cœurs multithreading).

4-2-1 Le modèle à mémoire partagée

Le modèle à mémoire partagée permet d'exprimer de manière simple et puissante le parallélisme dans une application. Dans ce modèle, plusieurs processus partagent un espace mémoire commun où ils peuvent lire et écrire des données de manière asynchrone.

Le modèle à mémoire partagée est implémenté dans une large variété de multiprocesseurs (**SMP** (Symmetrical Multiprocessor) et machines **NUMA**). Le regain d'intérêt pour les multiprocesseurs a poussé plusieurs constructeurs à définir un standard qui est **OpenMP**.

OpenMP est constitué d'un jeu de directives, d'une bibliothèque de fonctions et d'un ensemble de variables d'environnement **[21]**.

4-2-2 Le modèle à échange de messages

Dans ce modèle, un programme est constitué par un ensemble de processus opérant chacun dans un espace d'adressage privé. Les processus communiquent par échange de messages. Un système d'échange de messages demande un schéma d'adressage qui repère les processus les uns par rapport aux autres et deux primitives de base (Send/Receive) permettant le transfert de données entre processus. Ce modèle donne un contrôle total de la mise en œuvre du parallélisme au programmeur qui doit cependant gérer explicitement la distribution des données, les communications entre processus et les synchronisations. Le modèle à échange de messages est largement utilisé dans

les machines à mémoire distribuée mais il est aussi implémenté sur des machines à mémoire physiquement partagée (machines NUMA) avec de bonnes performances.

MPI (**M**essage **P**assing **I**nterface) et **PVM** (Parallel Virtual Machine) sont les bibliothèques les plus utilisées dans le modèle de programmation à échange de messages. Elles supportent les langages C, C++ et Fortran. Elles permettent d'écrire de manière simple et efficace des programmes parallèles s'exécutant sur n'importe quelle plateforme **[22]**.

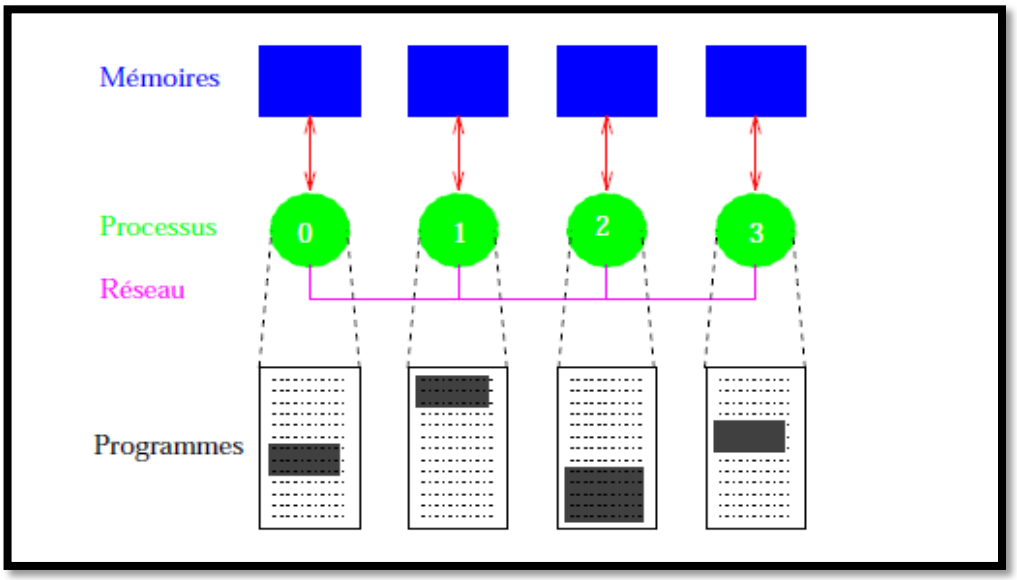


Figure III-3 : **Modèle de programmation par Echange de Messages**

4-2-3 Le modèle du parallélisme de données

Cet autre modèle exploite le parallélisme dérivant de l'application de la même opération à tous les éléments d'un ensemble de données. Les processus exécutent le même code mais ils opèrent sur des données différentes. Le problème est d'abord décomposé en plusieurs tâches de petite taille. Chaque tâche est ensuite assignée à un processeur qui effectue sur les données locales les opérations avant déterminer leur nouvelle valeur. Conceptuellement ce modèle est plus simple que le modèle à échange de messages. Les détails de la distribution des données et de la communication entre les processeurs sont masqués par le compilateur. Le programmeur doit cependant fournir le

maximum d'informations sur la distribution des données au compilateur afin de minimiser la communication entre les processus. Le parallélisme de données est utilisé aussi bien dans les systèmes à mémoire partagée que dans les systèmes à mémoire distribuée. Les implémentations de ce modèle sont souvent des extensions des langages de programmation séquentielle existants (Java, C++, Fortran, etc.).

HPF (High Performance Fortran) est une extension du langage Fortran. C'est le langage le plus utilisé pour écrire des programmes en utilisant le parallélisme de données. Il a été conçu pour réaliser la portabilité des programmes à travers différentes architectures parallèles tout en maintenant élevées leurs performances **[21]**.

6) Environnement MPI

5-1 Définition

Une application **MPI** (*Message Passing Interface*) est un ensemble de processus autonomes exécutant chacun leur propre code et communiquant via des appels à des sous-programmes de la bibliothèque MPI [**21].**

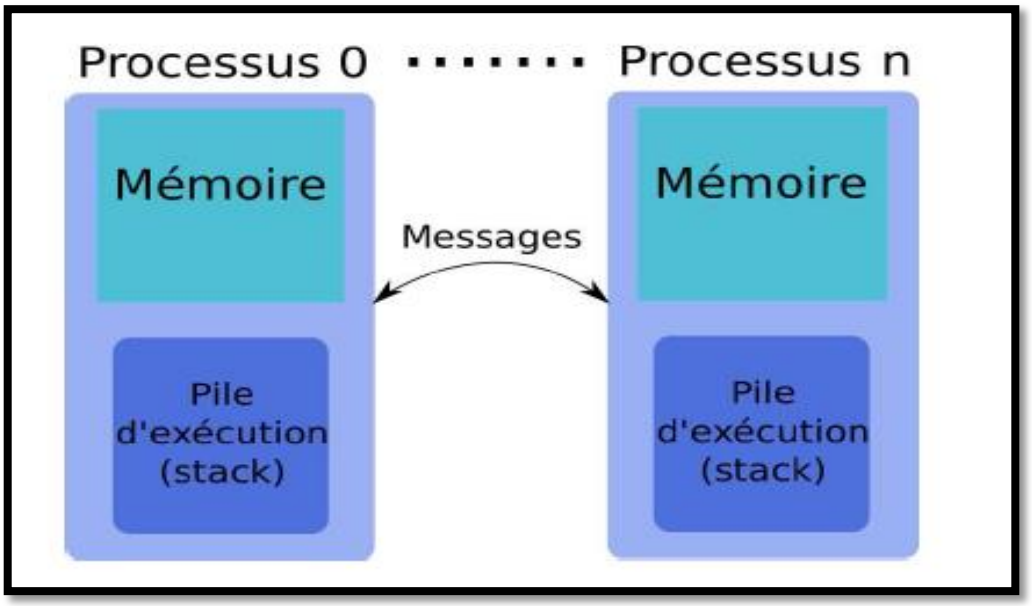


Figure III-4 : **Schéma générale MPI**

5-2 Description

Pour utiliser la bibliothèque MPI, le programme source doit impérativement contenir :

- L'appel au module MPI **:** include mpif.h en fortran77, use MPI en fortran90, include mpi.h en C/C + +.
- L'initialisation de l'environnement via l'appel MPI_INIT. Cette fonction retourne une valeur dans la variable code. Si l'initialisation s'est bien passée, la valeur de code est égale à celle dans MPI_SUCCESS.
- La désactivation de l'environnement via l'appel MPI_FINALIZE.

Une fois l'environnement MPI initialisé, on dispose d'un ensemble de processus actifs et d'un espace de communication au sein duquel on va pouvoir effectuer des opérations MPI. Ce couple (processus actifs, espace de communication) est appelé communicateur.

Le communicateur par défaut est MPI_COMM_WORLD et comprend tous les processus actifs. Il est initialisé lors de l'appel à la fonction MPI_INIT () et désactivé par l'appel à la fonction MPI_FINALIZE () **[23]**.

5-3 Les communications

On distingue deux types de communications sous MPI :

5-3-1 Les communications point à point [22] :

La communication point à point est une communication entre deux processus. L'un d'eux envoie un message (c'est l'émetteur, l'autre le reçoit (c'est le récepteur).

Ce message doit contenir un certain nombre d'informations pour assurer une bonne réception et interprétation par le récepteur, à savoir :

- Le rang du processus émetteur (rang_proc_source),
- Le rang du processus récepteur (rang_proc_dest),
- L'étiquette du message (tag_emis pour message émis, tag_recu pour message reçu),
- Le nom du communicateur (comm),
- Le type des données échangées (type_emis pour message émis, type_recu pour message reçu),

- Le nom données échangées (val_emis pour message émis, val_recu pour message reçu),
- La taille des données échangées (taille_emis pour message émis, taille_ reçu pour message reçu) (scalaire, vecteur, matrice, ...).

Plusieurs modes de transfert sont possibles pour échanger des messages. On décrit ici des fonctions MPI de communication en mode bloquant qui laisse la main une fois que le message est bien reçu. C'est le mode à utiliser quand on commence à paralléliser un code. On peut ensuite passer à un autre mode de transfert pour optimiser le temps de communication.

5-3-2 Les communications collectives [22]

Elles permettent de communiquer en un seul appel avec tous les processus d'un communicateur. Ce sont des fonctions bloquantes, c'est à dire que le système ne rend la main à un processus qu'une fois qu'il a terminé la tache collective.

Pour ce type de communication, les étiquettes sont automatiquement gérées par le système.

Dans ce type de communication on trouve des fonctions :

- **Diffusion générale : MPI_BCAST** *(val_emis, taille_emis, type_emis, rang_proc_source, comm, code)**:*** Cette fonction permet à un processus de diffuser (BCAST pour broadcast) un message à tous les processus du communicateur indiqué.

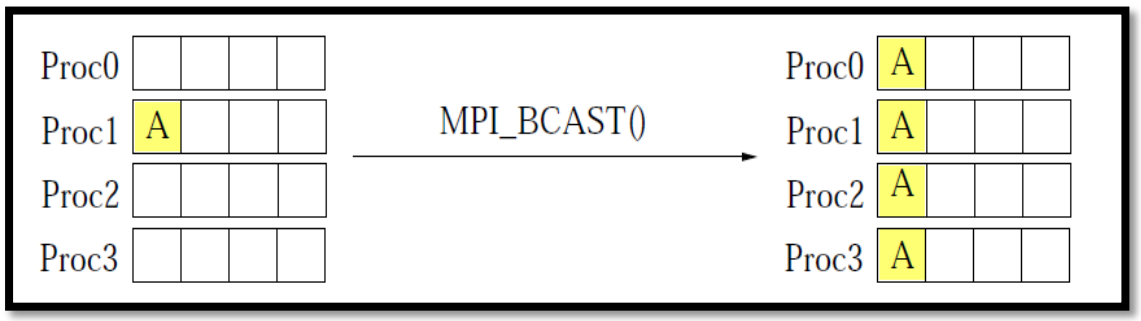


Figure III-5 : **Diffusion générale MPI_BCAST**

- **Diffusion sélective de données réparties : MPI_SCATTER** *(val_emis, taille_emis, type_emis, val_recu, taille_recu, type_recu, rang_proc_source, comm, code)* **:** Cette fonction permet à un processus de diffuser des données aux processus du communicateur indiqué de façon sélective.

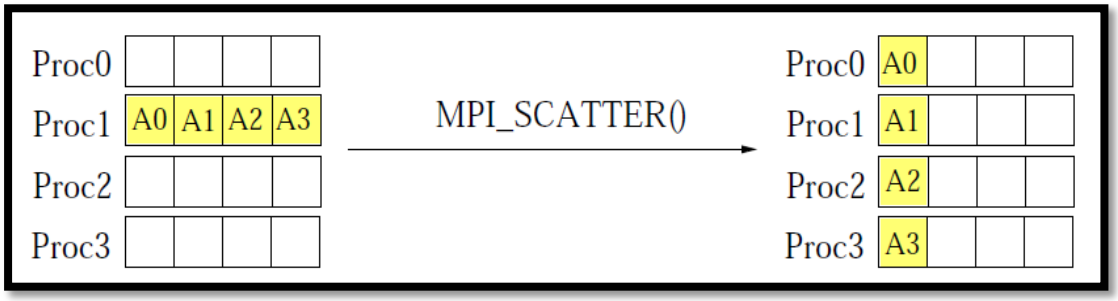


Figure III-6 : **Diffusion sélective : MPI_SCATTER**

- **Collecte de données réparties : MPI_GATHER** *(val_emis, taille_emis, type_emis, val_recu, taille_recu, type_recu, comm, code)* : Cette fonction permet au processus récepteur de collecter les données provenant de tous les processus (lui-même compris).

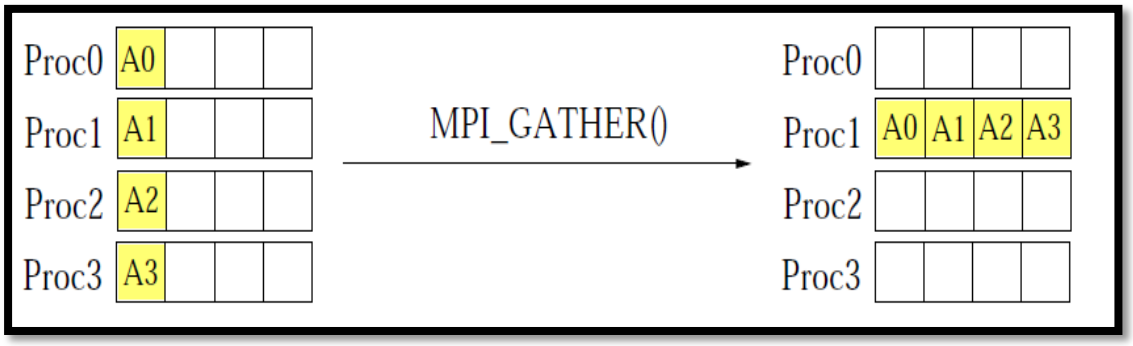


Figure III-7 : **Collecte de données réparties : MPI_GATHER**

- **Collecte Générale : MPI_ALLGATHER** *(val_emis, taille_emis, type_emis, val_recu, taille_recu, type_recu, rang_proc_dest, comm, code)* : Cette fonction effectue la même chose que la fonction MPI_GATHER, excepté que le résultat de la collecte est connue de tous les processus du communicateur.

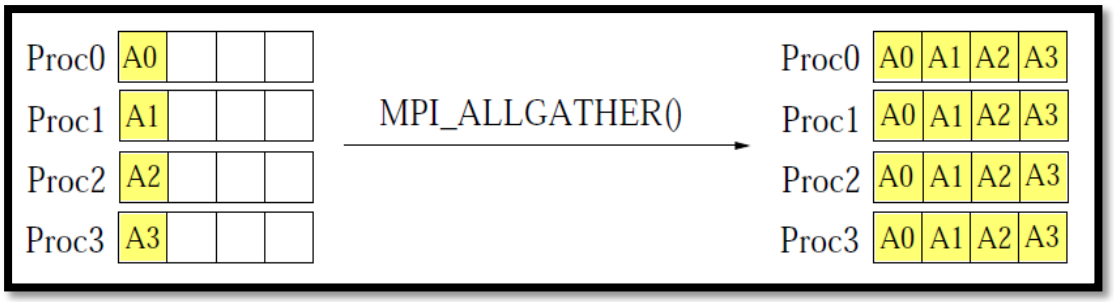


Figure III-8 : **Collecte générale : MPI_ALLGATHER**

- **Synchronisation globale : MPI_BARRIER** *(comm, code)* : Cette fonction bloque les processus à l'endroit où elle est appelée dans le programme. Les processus restent en attente au niveau de cette barrière (BARRIER) jusqu'à ce qu'ils y soient tous parvenus. Ensuite, ils sont libérés.

Il ya aussi d'autres fonction des opérations de réduction (MPI_SUM ; MPI_PROD ; MPI_MAX ; MPI_MIN), Une réduction consiste à appliquer une opération à un ensemble d'éléments pour obtenir un scalaire. Cela peut être la somme des éléments d'un vecteur ou la valeur maximale d'un vecteur.

- **MPI_REDUCE***(val_emis, val_recu, taille_emis, type_emis, operation, rang_proc_dest, comm, code)* : qui permet de faire des opérations de réduction sur des données réparties. Le résultat de l'opération de réduction est récupéré sur un seul processus.

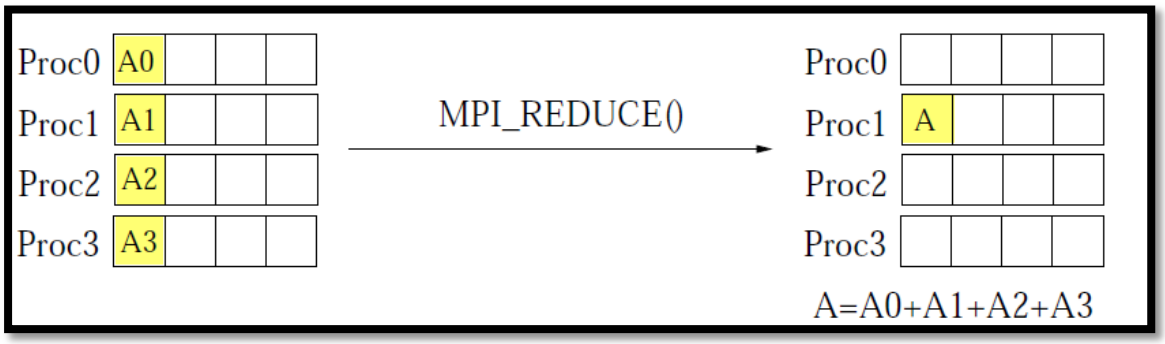


Figure III-9 : **Résultat de la fonction MPI_REDUCE**

- **MPI_ALL_REDUCE** *(val_emis, val_recu, taille_emis, type_emis, operation, comm, code)* : qui permet de faire les mêmes opérations de réduction que *MPI_REDUCE ()* **;** La différence est que le résultat de l'opération de réduction est connu de tous les processus d'un même communicateur.

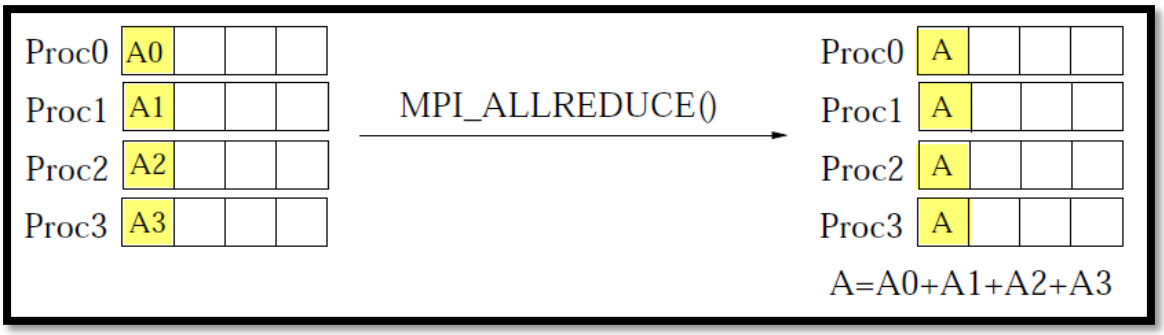


Figure III-10 : **Résultat de la fonction MPI_ALL_REDUCE**

Conclusion

Dans ce chapitre, nous avons présenté le principe du parallélisme et ces avantages en citant la classification des architectures parallèles, et les différents types d'architectures parallèles. Nous avons aussi expliqué la programmation parallèle avec ses différents modèles. Nous avons terminé ce chapitre par une étude détaillé du l'environnement **MPI** qui permet d'appliqué le principe du parallélisme en se basant sur le principe d'échanges de messages (communications) entre processus, chacun ayant sa propre mémoire (Machine a Mémoire distribuée). Ces opérations de communications peuvent être collectives ou point à point.

Introduction

Avec les progrès des technologies de collecte des données et de stockage, il est devenu inévitable de faire face au problème de calcul dans les systèmes distribués. Pour maintenir les performances de ces systèmes il faut faire l'équilibrage de charge.

Dans ce chapitre, nous proposons l'implémentation de la première partie de l'équilibrage charge dynamique dans un réseau local LAN.

La première étape : Estimation de la charge du groupe

1. Collecte périodique de l'information de charge

Pour chaque élément *Ei* de *G* **faire**

Envoi de sa charge actuelle *LODi* à son gestionnaire associé.

Fin Pour

L'implémentation de cet algorithme se base sur l'architecture à mémoire distribué qui nécessite au moins deux machines connectés (un serveur et un nœud) dans un réseau local avec un système d'exploitation *ubuntu linux* version **10.04.**

1) Etude Comparative

L'architecture à mémoire partagée et l'architecture à mémoire distribuée, peuvent être facilement comparées car elles sont en opposition sur un grand nombre de points.

Le choix entre les deux architectures est principalement posé sur le nombre de processeurs et les communications nécessaires entre eux. Peu de processeurs et peu de communication impliquent une mémoire partagée, En revanche beaucoup de processeur et de grands besoins de communication impliquent des mémoires distribuées.

Architecture à Mémoire Distribuée	*Architecture à Mémoire Partagée*
✓ Temps d'accès à la mémoire court	✓ Temps d'accès à la mémoire dépendant de la charge
✓ Grand nombre des processeurs	✓ Petit nombre des processeurs
✓ Communication spécifique	✓ Communication par boite à message
✓ Architecture flexible et scalable	✓ Architecture peu modifiable
✓ Architecture d'interconnexion spécifique	✓ Interconnexion par bus central
✓ Petites mémoires distribuées	✓ Grande mémoire centralisée

2) Configuration du Serveur

Avant tout, on doit avoir une adresse IP statique ou réservée en DHCP. Ensuite quelques paquets sont aussi nécessaires **[24]** :

- ✓ *libmpich1.0-dev.*
- ✓ *libmpich-mpd1.0-dev.*
- ✓ *libmpich-shmem1.0-dev.*
- ✓ *mpich2.*
- ✓ *mpich2-doc*
- ✓ *openssh-server.*
- ✓ *build-esentials.*

```
hemza@hemza-laptop:~$ sudo apt-get install libmpich1.0-dev libmpich-mpd1.0-dev
libmpich-shmem1.0-dev mpich2 mpich2-doc john openssh-server build-essential
```

Configuration du réseau : Un fichier nommé hosts dans le répertoire *etc* contient la définition de l'adresse IP de la machine et par défaut est égale à 127.0.1.1, on doit la remplacer par celle utilisée par la machine dans le réseau.

Configuration de l'utilisateur : on doit créer un utilisateur nommé Cluster, et ajouter le répertoire *bin* à son *path*.

```
hemza@hamza-laptop:~$ sudo useradd -m -s /bin/bash cluster
hemza@hamza-laptop:~$ sudo passwd cluster
Enter new UNIX password:
Retype new UNIX password:
passwd: password updated successfully
petur@server:~$ sudo su - cluster -c "mkdir ~/bin;export PATH=~/bin:$PATH"
```

Configuration de MPICH : Cette configuration nécessite la création d'un fichier de configuration *.mpd.conf* et un fichier *mpd.hosts* qui contient l'ensemble des machines autorisées dans notre cluster.

```
cluster@hamza-laptop:~$ touch ~/.mpd.conf
cluster@hamza-laptop:~$ chmod 600 ~/.mpd.conf
cluster@hamza-laptop:~$ echo secretword=pass>~/.mpd.conf
cluster@hamza-laptop:~$ /sbin/ifconfig|grep "inet addr"
inet addr:192.168.1.2 Bcast:192.168.1.255 Mask: 255.255.255.0
inet addr:127.0.0.1 Mask:255.0.0.0
cluster@hamza-laptop:~$ cat /proc/cpuinfo|grep processor|wc -l
4
cluster@hamza-laptop:~$ echo 192.168.1.2:1>~/mpd.hosts
```

3) Configuration des Nœuds

Cette étape se divise en deux parties, la première contient tout d'abord l'installation des packages nécessaires ensuite la configuration du réseau et de l'utilisateur, la deuxième contient la configuration de MPICH.

La première est la même que celle de la configuration du serveur, la deuxième doit être exécutée sur le serveur. On génère une clé RSA pour établir une connexion sécurisée entre les machines.

```
cluster@hamza-laptop:~$ ssh-keygen -t rsa
Generating public/private rsa key pair.
Enter file in which to save the key (/home/cluster/.ssh/id_rsa):
Created directory '/home/cluster/.ssh'
Enter passpharse (empty for no passpharse):
Enter same passpharse again:

Your identification has been saved in /home/cluster/.ssh/id_rsa.
Your public key has been saved in /home/cluster/.ssh/id_rsa.pub.
The key fingerprint is:
d1:71:52:9e:28:37:e1:c8:f0:b9:ea:82:45:3f:55:a4 cluster@hamza-laptop
The key's randomart image is:
+--[ RSA 2048]----+
|     . .=.o    |
|     + *.B .   |
|      E.* o    |
|   .  .= .     |
|  . . .S       |
|   . o.        |
|  o ..         |
| . ..          |
|    ..         |
+-----------------+

cluster@hamza-laptop:~$ ssh cluster@192.168.1.3 mkdir -p .ssh
cluster@192.168.1.3's password:

cluster@hamza-laptop:~$ cat .ssh/id_rsa.pub | ssh cluster@192.168.1.3
'cat>>.ssh/authorized_keys'
cluster@192.168.1.3's password:
cluster@hamza-laptop:~$ ssh cluster@192.168.1.3 'cat /proc/cpuinfo|grep
processor|wc -l'
4
cluster@hamza-laptop:~$ echo 192.168.1.3:4 >> ~/mpd.hosts
cluster@hamza-laptop:~$ for i in `cut --delimiter=: -f1 ~/mpd.hosts`;do scp
~/.mpd.conf cluster@$i:~;scp
~/mpd.hosts cluster@$i:~;done
The authenticity of host '192.168.1.2 (192.168.1.2)' can't be established.
RSA key fingerprint is 68:95:06:ae:af:e0:b4:7f:d3:c7:50:c9:b9:66:81:1f.
Are you sure you want to countinue connecting (yes/no)? Yes
Warning: Permanently added '192.168.1.2' (RSA) to the list of known hosts.
cluster@192.168.1.2's password:
.mpd.conf      100%     19         0.0KB/s         00:00
cluster@192.168.1.2's password:
mpd.hosts      100%     28         0.0KB/s         00:00
.mpd.conf      100%     19         0.0KB/s         00:00
mpd.hosts      100%     28         0.0KB/s         00:00
```

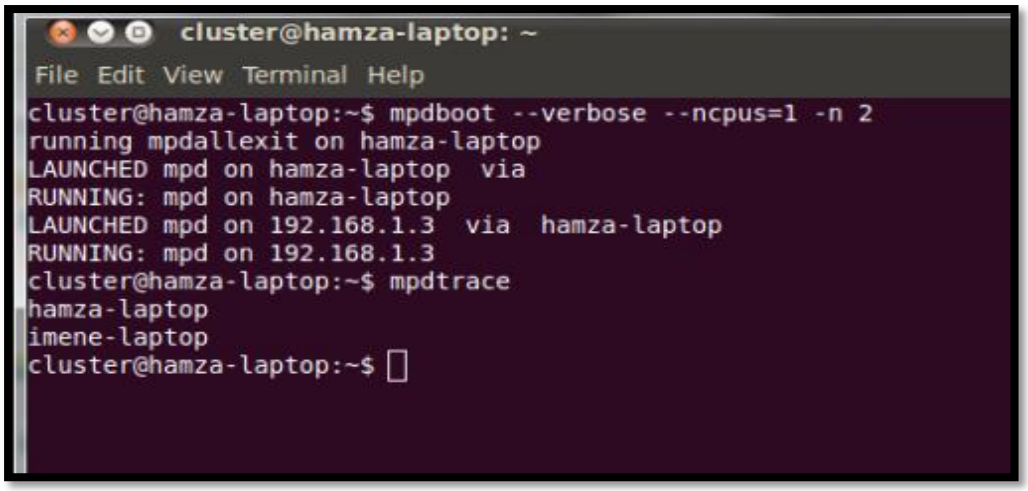


Figure VI.1 : **Lancement du cluster**

- **mpdboot :** pour lancer le cluster,
- **mpdtrace :** liste des nœuds de cluster,
- **mpdallexit :** fermer le cluster.

4) Echange de messages

La communication entre processus se fait uniquement par passage de messages entre processus (SEND-RECEIVE). Techniquement, cette communication se fait via des fonctions de la bibliothèque MPI appelées dans le programme. L'environnement MPI permet de gérer et interpréter ces messages.

send3.c

```
#include "mpi.h"
#include <stdlib.h>
#include <stdio.h>
#define ETIQUETTE 100
#define max 12

int main(int argc, char* argv[])
{
int rank, nb ;
MPI_Status status;
MPI_Init (&argc,&argv);
MPI_Comm_rank ( MPI_COMM_WORLD,&rank);
if (rank == 0) {
for (nb=0; nb<max; nb++)
MPI_Send (&nb, 1, MPI_INT , 1, ETIQUETTE, MPI_COMM_WORLD );
}
else if (rank == 1) {
MPI_Recv (&nb, 1, MPI_INT , 0, ETIQUETTE, MPI_COMM_WORLD, &status);
for (nb=0; nb<max; nb++)
if ((nb % 2 )== 0)
printf("nombre pair : %d.\n", nb);
}
MPI_Finalize ();
return(0);
}
```

Figure VI.2 : **Exemple de programme en C (Affiche les processus pair)**

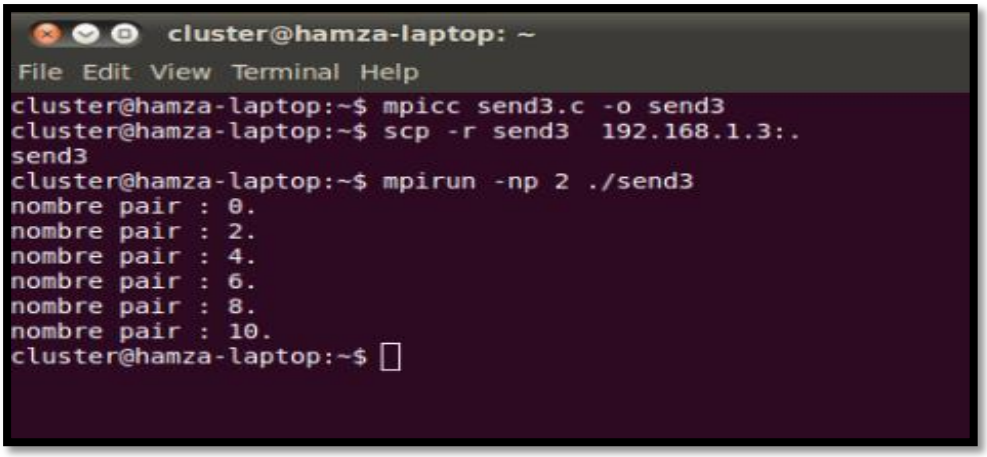


Figure VI.3 : **Exécution de l'exemple précédant**

5) L'application

Après la configuration de cluster, Nous avons appliqué la première étape du l'algorithme d'équilibrage de charge l'algorithme **« Estimation de la charge »** qui nécessite de calculer la charge de travail des nœuds et de transférer ces valeurs vers le serveur. A cet effet, Nous avons commencé par l'estimation du temps et la charge actuelle du système de chaque machine. Les résultats obtenus seront enregistrés dans un fichier *« time.txt »* (contient le temps) et dans un autre fichier *«free.txt»* (contient la capacité mémoire) et enfin dans un fichier *« «cpu.txt »* qui contient la capacité de Cpu.

fichier2.c | free.txt | fichier3.c | time.txt

```c
#include "mpi.h"
#include <stdlib.h>
#include <stdio.h>
int main(int argc, char* argv[])
{
int rank;
MPI_Init (&argc,&argv);                    /* initialiser l'environement MPI */
MPI_Comm_rank ( MPI_COMM_WORLD,&rank);     /* le communicateur */
FILE* fichier=NULL;
fichier = fopen("time.txt","r");           /* ouvrir un fichier */

if (rank == 0)
{
system("uptime >> time.txt");              /* commande uptime pour calculer le temp */
printf("l'excution le temp du noeud ");    /* l'affichage */
system("cat time.txt");                    /* pour mettre les resultat dans un fichier */
}
 else if (rank == 1)
{
system("uptime >> time.txt");
printf("l'excution le temp du serveur ");
system("cat time.txt");
//system("free");
}

MPI_Finalize ();                           /*desactiver l'environement MPI*/

fclose(fichier);                           /*fermer le fichier*/
return (0);
}
```

Figure VI.4 : **programme en C qui estimer le temps « fichier3.c »**

```
cluster@hamza-laptop:~$ mpicc fichier3.c -o fichier3
fichier3.c: In function 'main':
fichier3.c:14: warning: ignoring return value of 'system', declared with attribu
te warn_unused_result
fichier3.c:16: warning: ignoring return value of 'system', declared with attribu
te warn_unused_result
fichier3.c:20: warning: ignoring return value of 'system', declared with attribu
te warn_unused_result
fichier3.c:22: warning: ignoring return value of 'system', declared with attribu
te warn_unused_result
cluster@hamza-laptop:~$ scp -r time.txt  192.168.1.3:.
time.txt                                      100%   21     0.0KB/s   00:00
cluster@hamza-laptop:~$ mpirun -np 2 ./fichier3
l'excution le temp du noeud l'excution le temp du serveur le temp de cluster


 02:20:16 up 10 min,  2 users,  load average: 0.16, 0.17, 0.11
le temp de cluster


 01:20:18 up 12:27,  2 users,  load average: 0.24, 0.30, 0.20
cluster@hamza-laptop:~$
```

Figure VI.5 : **Exécution de l'exemple précédant « fichier3.c »**

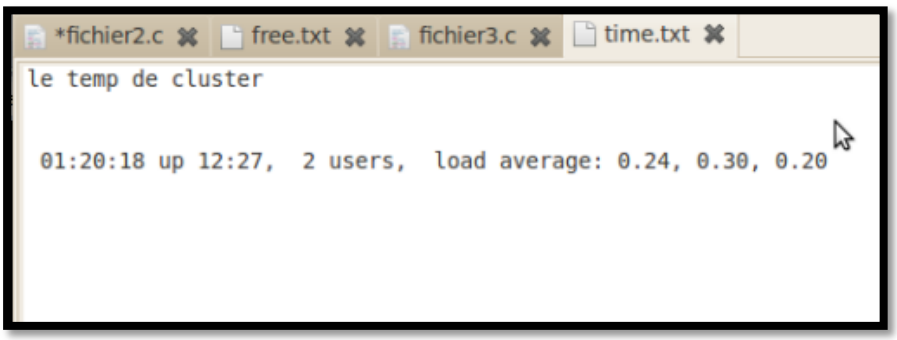


Figure VI.6 : **L'enregistrement des résultats de Serveur dans le fichier « time.txt »**

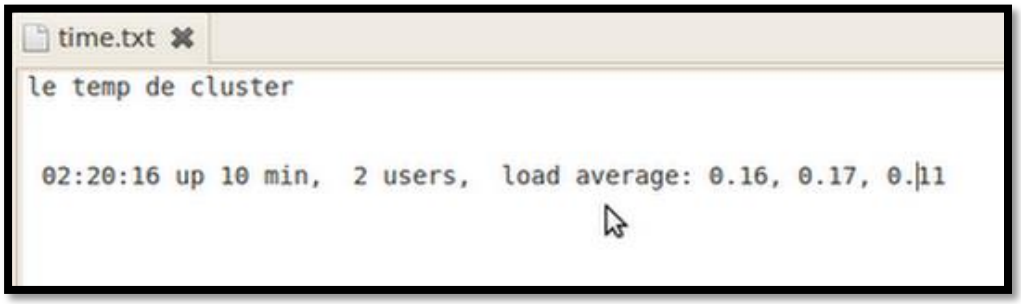


Figure VI.7 : **L'enregistrement des résultats du Nœud dans le fichier « time.txt »**

Le programme **« fichier3.c »** affiche sur une ligne les informations suivantes pour chaque machine (serveur ou nœud) :

- L'heure actuelle ;
- Le temps depuis lequel le système est en marche ;
- Le nombre d'utilisateurs connectés (2 Users) ;
- La charge du système.

Ci-dessus le deuxième programme qui sert a estimé la disponibilité de la mémoire vive des deux machines en utilisant la commande système *« free »*.

fichier2.c | free.txt | fichier3.c | time.txt

```
#include "mpi.h"
#include <stdlib.h>
#include <stdio.h>
int main(int argc, char* argv[])
{
int rank;
MPI_Init (&argc,&argv);                    /* initialiser l'environement MPI */
MPI_Comm_rank ( MPI_COMM_WORLD,&rank);     /* le communicateur */
FILE* fichier=NULL;
fichier = fopen("free.txt","r");           /* ouvrir un fichier */

if (rank == 0)
{
system("free >> free.txt");                /* commande free pour calculer la charge */
printf("l'excution la charge du noeud \n "); /* l'affichage */
system("cat free.txt");                    /* pour mettre les resultat dans un fichier */
//system("free");
}
else if (rank == 1)
{
system("free >> free.txt");
printf("l'excution la charge du serveur\n");
system("cat free.txt");
//system("free");
}
MPI_Finalize ();                           /*desactiver l'environement MPI*/

fclose(fichier);                           /*fermer le fichier*/
return (0);
}
```

Figure VI.8 : **Programme en c qui estime l'espace mémoire « fichier2.c »**

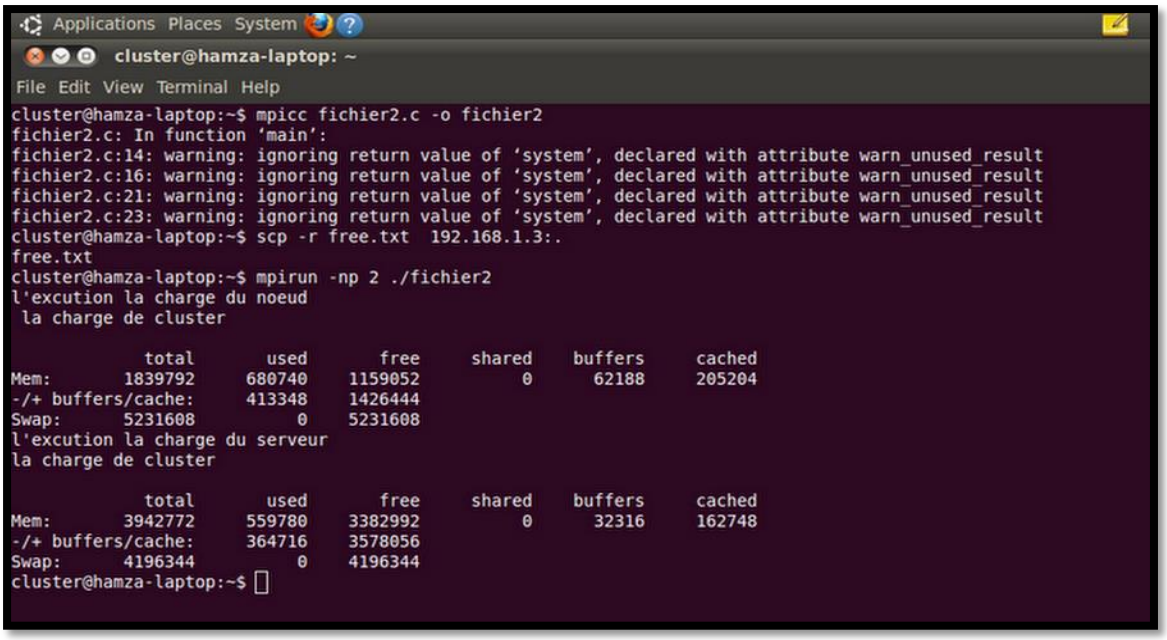


Figure VI.9 : **Exécution de l'exemple précédant « fichier2.c »**

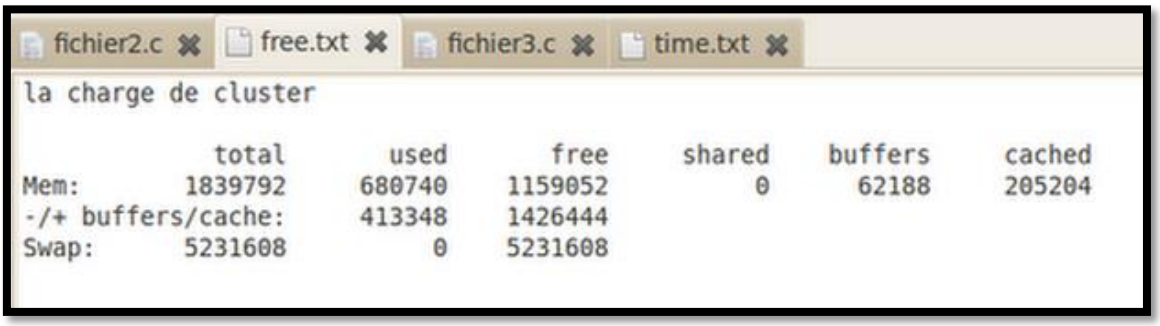


Figure VI.10 : **l'enregistrement des résultats du serveur dans**

Le fichier « free.txt »

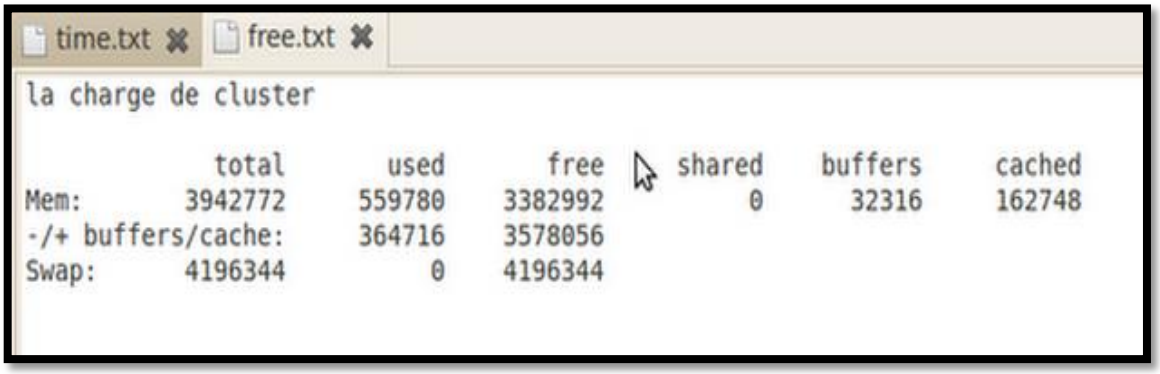


Figure VI.11 : **l'enregistrement des résultats du Nœud dans**

le fichier « free.txt »

Le programme **« fichier2.c »** affiche sur trois lignes les informations suivantes pour chaque machine (serveur ou nœud) :

Ligne 1 : Indique les détails sur l'espace mémoire (l'espace du mémoire totale, utilisé et libre, partagée RAM, RAM utilisée pour les tampons, RAM utilisée de contenu de cache).

Ligne 2 : Indique l'espace du mémoire tampon (buffers) et la mémoire cachée (cached).

Ligne 3 : Indique la mémoire totale de swap disponible, échange utilisée et échange libre de la mémoire disponible.

Ci-dessus le troisième programme qui sert a estimé la disponibilité de la mémoire vive ainsi la capacité de cpu des deux machines en utilisant la commande système *« free » et « cpu info»*

```
#include <stdlib.h>
#include <stdio.h>
int main(int argc, char* argv[])
{int rank;
MPI_Init (&argc,&argv);                                   /* initialiser l'environement MPI */
MPI_Comm_rank ( MPI_COMM_WORLD,&rank);                    /* le communicateur */
FILE* fichier=NULL;
//fichier = fopen("free.txt","w");
//printf(fichier,"l'excution 1\n ");
if (rank == 0)
{
fichier = fopen("cpu.txt","w");                           /* ouvrir un fichier */
printf(fichier,"l'excution de la charge\n ");
system("cat /proc/cpuinfo | grep model | cut -c14- >> cpu.txt");   /* commande cpuinfo pour donner la capacité cpu */
printf("l'excution de serveur\n ");
system("free >> cpu.txt");                                /* commande free pour calculer la charge */
system("cat cpu.txt");                                    /* pour mettre les resultat dans un fichier */
}
else if (rank == 1)
{
fichier = fopen("cpu.txt","w");                           /* ouvrir un fichier */
printf(fichier,"l'excution de la charge\n ");
system("cat /proc/cpuinfo | grep model | cut -c14- >> cpu.txt");   /* commande cpuinfo pour donner la capacité cpu */
printf("l'excution de noeud\n");
system("free >> cpu.txt");                                /* commande free pour calculer la charge */
system("cat cpu.txt");                                    /* pour mettre les resultat dans un fichier */
}MPI_Finalize ();                                          /*desactiver l'environement MPI*/

fclose(fichier);                                          /*fermer le fichier*/
return (0);
}
```

Figure VI.12 : **Programme en c qui estime l'espace mémoire + la capacité CPU**

```
cluster@hamza-laptop:~$ mpirun -np 2 ./fichier5
0s000B[0s000[1'excution de serveur
 l'excution de noeud

Intel(R) Core(TM) i3-2348M CPU @ 2.30GHz

Intel(R) Core(TM) i3-2348M CPU @ 2.30GHz

Intel(R) Core(TM) i3-2348M CPU @ 2.30GHz

Intel(R) Core(TM) i3-2348M CPU @ 2.30GHz
             total       used       free     shared    buffers     cached
Mem:       1839792     542188    1297604          0      32696     169996
-/+ buffers/cache:     339496    1500296
Swap:      5231608          0    5231608

Intel(R) Core(TM) i3-3217U CPU @ 1.80GHz

Intel(R) Core(TM) i3-3217U CPU @ 1.80GHz

Intel(R) Core(TM) i3-3217U CPU @ 1.80GHz

Intel(R) Core(TM) i3-3217U CPU @ 1.80GHz
             total       used       free     shared    buffers     cached
Mem:       3942772     677344    3265428          0      34796     224244
-/+ buffers/cache:     418304    3524468
```

Figure VI.13 : **Exécution de l'exemple précédant « fichier4.c »**

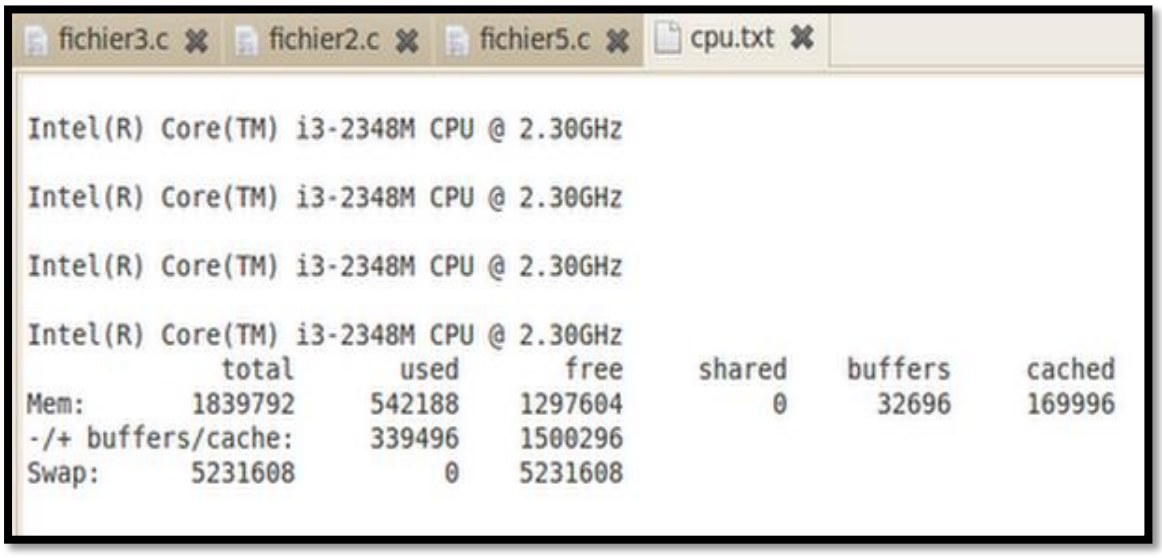


Figure VI.14 : **l'enregistrement des résultats du serveur dans**

Le fichier « cpu.txt »

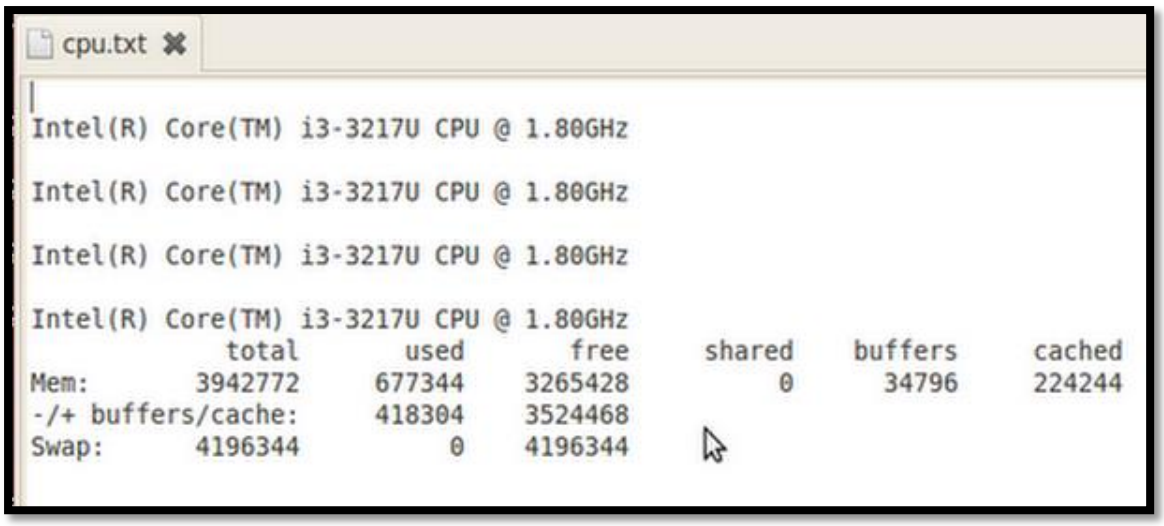


Figure VI.15 : **l'enregistrement des résultats du nœud dans le fichier « cpu.txt »**

Conclusion

Dans ce chapitre, nous avons proposé d'implémenter la première partie de l'équilibrage de charge dans un réseau local à savoir l'estimation de la charge de travail. A cette fin, nous avons proposé un modèle d'échange des messages entre deux machines reliées entre eux par un câble. Chaque ordinateur collecte son information de charge (Charge du système, Espace mémoire, capacité cpu) et l'envoie à l'autre ordinateur par l'utilisation de la bibliothèque MPI.

Nous avons donc réussi à implémenter l'étape la plus difficile de l'équilibrage de charge. Cependant il reste deux étapes dans l'équilibrage de charge à savoir la prise de décision et le transfert des tâches.

Conclusion Générale et Perspectives :

En informatique, l'équilibrage de charge (Load Balancing) est un ensemble de techniques permettant de distribuer une charge de travail entre différents ordinateurs d'un groupe, Cette pratique est devenue indispensable pour l'exploitation optimale des systèmes distribués.

Le problème de l'équilibrage étant un problème relativement ancien, beaucoup d'approches ont été proposées pour le résoudre. Nous avons intéressé à l'approche dynamique, où l'assignation des tâches aux machines se décide durant la phase d'exécution, en fonction des informations qui sont collectées sur l'état de charge du système. Ceci permet d'améliorer les performances d'exécution des tâches mais au prix d'une complexité dans la mise en œuvre de cette stratégie, notamment en ce qui concerne la définition de l'état de charge du système, qui doit se faire de manière continue.

La plus grande difficulté réside dans l'estimation de la charge globale de système. Ainsi, tout système d'équilibrage de charge devra, en premier lieu, permettre d'estimer l'état de charge de chaque ressource de calcul.

Dans ce mémoire, nous nous sommes intéressés à la technique de calcul de charge des ressources de calcul. A cette fin, nous avons proposé un modèle d'échange des messages entre deux machines à travers un réseau local LAN. Chaque ordinateur collecte son information de charge (Processeur, Mémoire) et l'envoie à l'autre ordinateur par l'utilisation de la bibliothèque MPI.

Comme perspectives, Nous envisageons d'implémenter les autres étapes de l'équilibrage de charge dans un réseau local, à savoir la prise de décision et le transfert des tâches.

Références Bibliographiques :

[1] : Modèles et Approches Formels pour les Systèmes Distribués - M.Mosbah.

[2] : Régulation de charge dans les systèmes distribués : architecture et algorithmes - Frédéric Wautelet.

[3] : Contributions aux techniques d'ordonnancement sur plates-formes parallèles ou distribuées - Par Lamiel Toch.

[4] : Migration Alpha – Itanium sous OpenVSM - Auteurs : ABOUNADA Sami, BOUKADIDA Mohamed J, KSOURI Sonia.

[5] : Interface de communication pour les réseaux InfiniBand : Projet de fin d'étude présenté par : Aïchatou RABBA - Nadir GHEZALI.

[6] : Le projet européen de grille de calcul (DataGrid) : Concept de grilles de calcul. CHARRON Thierry.

[7] : Sécurité dans les grilles de calcul - Réalisé par Jamal Eddine GHAFFOUR.

[8] : L'informatique en nuage IP/A/IMCO/ST/2011-18 Mai 2012.

[9] : Analyses et Synthèses – Les risques associés au Cloud Computing – Autorité du Control Prudentiel (ACP – Banque de France).

[10] : Comparaison et mise en place des plateformes de Cloud Computing : OpenStack et Eucalyptus : Mémoire MASTER ACADEMIQUE - Présenté par : AOUAMEUR Zakia, TAHRINE Halima - Université Kasdi Merbah de Ouargla.

[11] : Gestion de tolérance aux fautes dans les Cloud Computing avec CloudSIM – Mémoire pour l'obtention d'un mémoire de magister présenté par : Mr.LILAM Said.

[12] : Load Balancing auteurs : MASSAOUDI MOHAMED, CHAHINEZ HACHAICHI, AMENI DHAWEFI, ERIJ MAIJED, EMNA BOUGHANMI

[13] : Modèle d'Équilibrage de Charge pour les Grilles de Calcul auteur : YAGOUBI Belabbas.

[14] : Equilibrage de charge pour les grilles de calcul : classe des tâches dépendantes et indépendantes. Auteurs : Meriem Meddeber et Belabbas Yagoubi .

[15] : Algorithme de diffusion génétique pour l'équilibrage de charge dans les grilles de calcul. Auteur : DRIF Ahlem.

[16] : Modèle Arborescent pour l'Equilibrage de Charge dans les Grilles de Calcul - YAGOUBI Belabbas - Université d'Oran Es Sénia .

[17] : Load Balancing Task Scheduling based on Genetic Algorithm in Cloud Computing Tingting Wang, ZhaobinLiu , Yi Chen, Yujie Xu .

[18] : ANET : un environnement de programmation a parallélisme de données pour l'analyse d'image, Thèse par Nicolas Sicard - le 12 juillet 2004.

[19] : Contribution à la parallélisation des algorithmes exacts de résolution de contraintes, Thèse de Magister, présentée et soutenue publiquement par SAIDI Samira- 2013.

[20] : Parallélisme, Notes de cours – Octobre 2004, Cours de Bastien Chopard.

[21] : Message Passing Interface (MPI), Isabelle Dupays - Marie Flé Jérémie Gaidamour , Dimitri Lecas – Juin 2014 .

[22] : Introduction au calcul parallèle avec la bibliothèque MPI (Message Passing Interface), Stéphanie DELAGE SANTACREU – Novembre 2008.

[23] : Interrogation des bases de données dans le cluster sans SGBD, DOUZI Salah eddine, 2012-2013.

[24] : John the Ripper on a Ubuntu 10.04 MPI Cluster. Pétur Ingi Egilsson petur [at] petur [.] eu 2010.

Liste des figures

Résumé

La distribution des calculs sur plate-forme parallèle et distribuée peut être effectuée dynamiquement, ce qui nécessite des phases périodique de réallocation et de communication pour remédier au déséquilibre de charge.

Dans un premier temps, nous nous sommes intéressés à l'équilibrage de charge sur plate-forme Grille de calcul. Nous sommes tout d'abord placé un réseau local (LAN) entre deux ordinateurs, composés de processeurs ayant des vitesses de calcul différentes. Chaque ordinateur collecte son information de charge (Processeur, Mémoire) et l'envoie à l'autre ordinateur en utilisant la technique de passage de messages MPI.

Les Mots Clés : Système distribué ; Equilibrage de charge ; Les Grilles Informatiques ; Programmation parallèle.

Abstract

Distribution on parallel and distributed platform calculations can be performed dynamically, requiring periodic phases reallocation and communication to address the unbalanced load.

Initially, we were interested in load balancing platform Computing Grid. We are first placed a local network (LAN) between two computers, consisting of processors with different computing speeds. Each computer collects the load information (CPU, Memory) and sends it to another computer using the MPI message passing technique.

Keywords: Distributed System; Load Balancing; Grid Computing; parallel programming.

مـلخص

إن توزيـع الحسـابات علـى المنصـات الموازيـة يمكـن أن تجـرى بشـكل حيـوي، والتـي تتطلـب مراحـل دوريـة لإعادة التوزيع والاتصالات لمعالجة مشكل الحمولة غير المتوازنة.

فـي البدايـة، كنـا مهتمـين موازنـة الحمولـة علـى مسـتوى الحوسـبة الشـبكية. فـي أول الأمـر قمنـا بإنشـاء شـبكة محليـة **(LAN)** بـين جهـازي كمبيـوتر يتكـون كـل واحـد منهمـا مـن معالجـات بسـرعات مختلفـة. كـل كمبيـوتر بجمـع المعلومـات (سـرعة المعـالج، الـذاكرة) ويرسـله إلـى كمبيـوتر آخـر باسـتخدام تقنيـة تبـادل الرسـائل **MPI**.

الكلمات المفتاحية : النظام الموزع ، موازنة الحمولة ، الحوسبة الشبكية ، البرمجة المتوازية.