UNIVERSITE MOULOUD MAMMERI DE TIZI-OUZOU

Faculté de Génie électrique et d'Informatique

Département d'Informatique

Projet de fin d'études

En vue de l'obtention du diplôme de Master en Informatique

Option : Réseaux, mobilité et systèmes embarqués

Ordonnancement de tâches dépendantes sur la grille de calcul

Réalisé par:

OULD FELLA Mériem

TOUAHIR Fazia

Encadré par :

Mme K. OUKFIF

Promotion 2012/2013

Chapitre I: Les systèmes distribués à grande échelle

Chapitre II: Ordonnancement de tâches sur une grille de calcul

Chapitre III : Framework SimGrid

Chapitre IV: Implémentation

A l'ère des nouvelles technologies, l'homme ne cesse de fabriquer des machines très puissantes pour répondre aux besoins des applications de plus en plus gourmandes en puissance de calcul et en espace de stockage.

Des centres de calcul sont mis en œuvre et interconnectés afin de former des systèmes distribués plus grand. Ainsi, des environnements distribués à grandes échelles, comme les grilles informatiques et le nuage de calcul sont apparus.

Cependant l'exploitation de telles infrastructures (grille, cloud) n'est pas toujours facile vu les caractéristique des ressources qui les constituent. Parmi ces difficultés, on cite le déploiement d'une application distribuée, la tolérance aux pannes, le problème de transfert de données et l'ordonnancement des tâches indépendantes et dépendantes sur une grille de calcul. Nous nous sommes intéressées à ce dernier type des tâches.

Généralement, les tâches dépendantes d'une même application sont représentées sous forme d'un graphe (DAG pour Direct Acyclic Graphe), formant ainsi un workflow qui est une structure abstraite d'une application décrite par un DAG

En étudiant les principaux algorithmes d'ordonnancements pour ces applications, on s'est intéressé à deux algorithmes très connus Min-Min et HEFT (Heterogenous Eearlist Finish Time). Notre but est d'implémenter l'heuristique HEFT ; ensuite faire une étude comparative entre l'algorithme Min-Min et HEFT via des simulations réalisées sous SimGrid.

Le reste du mémoire est organisé ainsi, nous présentons un ensemble de concepts et définitions reliés aux systèmes distribués à grande échelle. .

Ensuite, nous présentons le problème d'ordonnancements et les différents algorithmes attribués pour ce problème dans le chapitre 2.

Dans le chapitre3 nous présentons le Framework SimGrid, son architecture et ses principales fonctions.

Dans le chapitre 4 nous présentons l'implémentation de l'algorithme HEFT et une étude comparative du temps de simulation obtenu avec Min-Min et HEFT. Enfin nous avons conclu par une conclusion et les perspectives.

1. Introduction

La disponibilité simultanée de nombreuse ressources de calcul/ stockage performantes,et un réseau de communication mondial suffisamment efficace et stable, pour véhiculer rapidement des grands volumes d'informations nécessitant des très grandes capacités de calcul et de stockage a donné naissance au concept de *Système Distribués à Grande Echelle* (SDGE). Au cours de ces dix dernières années, les systèmes distribués ont subi un changement d'échelle considérable, aujourd'hui, au travers de réseaux locaux (LAN) et planétaires (WAN) un très grand nombre d'ordinateurs sont interconnectés dans le monde [**1.1].**

2. Les systèmes distribués [1.2]

Un système informatique distribué est une collection de postes ou calculateurs autonomes qui sont connectés à l'aide d'un réseau de communication. Chaque poste exécute des composantes, par exemple des séquences de calculs, issues du découpage d'un projet de calcul global, et utilise un middleware, qui s'occupe d'activer des composantes et de coordonner leurs activités de telle sorte qu'un utilisateur perçoive le système comme un unique système intégré.

Une propriété importante des systèmes distribués est que la distribution est généralement cachée pour l'utilisateur. Il préfère voir l'ensemble comme un seul et unique système et ainsi cacher la complexité de la distribution le plus possible et augmenter la transparence du système distribué. Cela permet de développer le plus possible les applications de la même façon que les systèmes centralisés.

Un système distribué est généralement séparable en plusieurs composantes entièrement autonomes. Il n'existe pas de composante maître qui gère les autres et chacune est donc responsable de son propre fonctionnement. Cela permet, entre autres, d'avoir une hétérogénéité dans la technologie utilisée pour chaque composante, ils peuvent être écrits dans différents langages de programmation (Java, Cobol, C++, etc.) et s'exécuter sur différents Systèmes d'exploitation (Mac OS X, Linux, Windows, etc.).

La **figure1.1** représente un système distribué.

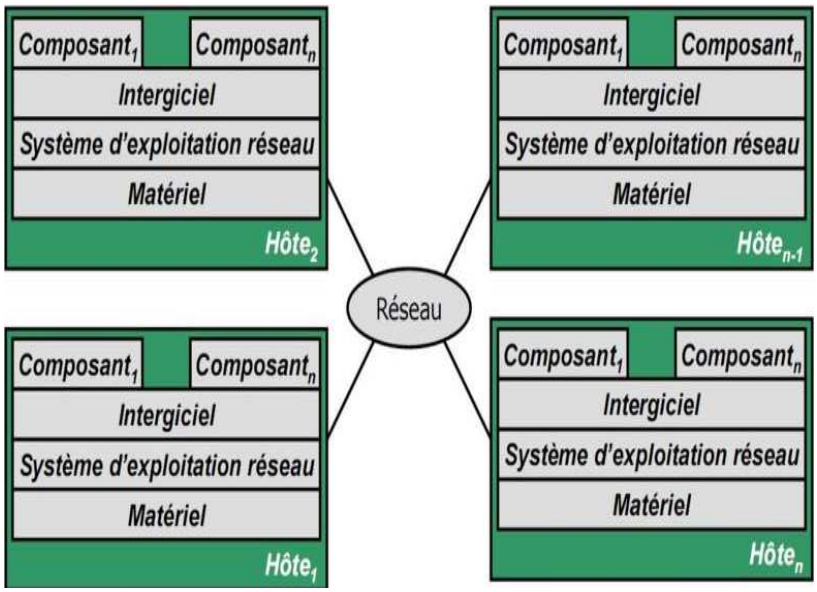


Figure 1.1 : Modèle d'un système distribué [1.3].

2.1. Objectifs d'un système distribué [1.3]

Un système distribué doit assurer plusieurs propriétés pour être considéré comme un système performant. On a cité cinq propriétés importantes pour un système distribué : Transparence, disponibilité, mise à l'échelle, tolérance aux pannes, la sécurité.

1) Transparence en terme de

- *Accès :* L'utilisateur ne se soucie pas du format de représentation des données.
- *Localisation :* L'utilisateur ne connaît pas où sont situées les ressources.
- *migration :* une ressource peut changer d'emplacement sans que cela ne soit aperçu
- *re-localisation* : cacher le fait qu'une ressource peut changer d'emplacement au moment où elle est utilisée.
- *Duplication :* L'utilisateur ne connaît pas le nombre de copies existantes.
- *Concurrence :* rendre invisible le fait qu'une ressource peut être partagée ou sollicitée simultanément par plusieurs utilisateurs.

- *panne* : si un nœud est en panne, l'utilisateur ne doit pas s'en rendre compte et encore moins de sa reprise après panne ;
- *Parallélisme :* Des tâches peuvent s'exécuter en parallèle.

2) Disponibilité

Un système est dit disponible s'il est en mesure de délivrer correctement le ou les services de manière conforme à sa spécification.

3) Passage à l'échelle (scalability)

Le concept de passage à l'échelle désigne la capacité d'un système de continuer à délivrer avec un temps de réponse constant un service même si le nombre de clients ou de données augmente de manière importante. Le passage à l'échelle peut être mesuré avec au moins trois dimensions :

i. Le nombre d'utilisateurs et/ou de processus (passage à l'échelle en taille) ;
ii. La distance maximale physique qui sépare les nœuds ou ressources du système (passage à l'échelle géographique) ;
iii. Le nombre de domaines administratifs (passage à l'échelle administrative). Le dernier type de passage à l'échelle est d'une importance capitale dans les grilles informatiques car il influe sur le degré d'hétérogénéité et donc sur la complexité du système

4) Tolérance aux pannes

- Détection de pannes (difficulté et même impossibilité de détection pour certains systèmes, suspicion de machines)
- Correction d'erreurs (de fichiers/messages corrompus)
- Reprise sur pannes (techniques de journalisation dans les BD) (éventuellement système dégradé).

5) Sécurité

- Confidentialité (authentification)
- Intégrité (protection contre les falsifications et les corruptions)
- Disponibilité (accès aux ressources)

2.2. Classification des systèmes distribués [1.4]

Les systèmes distribués peuvent être classés de différentes manières, trois classes ont été essentiellement identifiées à savoir les systèmes de calcul distribué (Distributed Computing Systems), les systèmes d'information distribués (Distributed Information Systems) et les systèmes pervasifs distribués (Distributed Pervasive Systems). Cette classification est axée essentiellement sur les domaines applicatifs des systèmes distribués plutôt que sur leur organisation interne (répartition géographique et support de communication) et les spécificités des ressources (hétérogénéité, stabilité des ressources, etc.). Pourtant, l'organisation et les caractéristiques des ressources sont des éléments essentiels qui permettent de bien prendre en compte les besoins spécifiques des applications en termes de performance. Ce faisant, nous avons fait un classement qui s'appuie plus sur l'organisation et les caractéristiques des ressources.

- ✓ *La grappe d'ordinateur (Cluster) :* c'est un ensemble d'ordinateurs connectés par un réseau LAN rapide et fiable pour assurer la disponibilité. Les ressources quasi-homogènes (même système d'exploitation, logiciels quasi-similaires) sont sous le contrôle d'un seul nœud appelé nœud maître.
- ✓ *La grille informatique (Grid) :* c'est une collection d'ordinateurs hétérogènes réparties sur différents sites maintenus par plusieurs organisations. Les sites sont reliés par un réseau WAN et chacun d'entre eux contient plusieurs ressources informatiques administrées de manière autonome et uniforme.
- ✓ *Les systèmes pair-à-pair (Peer-to-Peer (P2P))* : c'est un ensemble d'ordinateurs appelés pairs, qui s'accordent à exécuter une tâche particulière. Les ressources peuvent être des ordinateurs ou des assistants personnels interconnectés via

Internet, ce qui rend le système beaucoup plus flexible mais aussi moins fiable et stable.

- ✓ *Cloud :* du point de vue système, le Cloud est un ensemble d'ordinateurs (ressources) stockés sur de vastes grilles de serveurs ou de data centres. Les ressources du Cloud sont mutualisées à travers une virtualisation qui favorise la montée en charge, la haute disponibilité et un plan de reprise à moindre coût.

.

2.3. Avantages et inconvénients des systèmes distribués [1.5]

2.3.1. Avantages

Mise en commun d'un grand nombre de ressources à faible coûts :

- ✓ Bon rapport performance/prix
- ✓ Puissance globale virtuellement illimitée

Disponibilité et flexibilité :

- ✓ Un élément peut tomber en panne sans bloquer tout le système
- ✓ Distribution de la charge

Réponse à la distribution géographique de certains systèmes existants :

- ✓ Réplication locale des ressources globales

Partage de ressources coûteuses entre plusieurs utilisateurs/machines :

- ✓ Accès à distance à une ressource indisponible en local
- ✓ Accès aux mêmes ressources depuis tous les endroits du système

2.3.2. Inconvénients

Logiciels de gestion difficiles à concevoir ***:***

- ✓ Peu d'expérience ou succès dans ce domaine
- ✓ Complexité imposée par la transparence

Problèmes inhérents aux communications :

- ✓ Lenteur, saturation, perte de messages

Partage et distribution de données impose des mécanismes complexes :

- ✓ Synchronisation
- ✓ Sécurité

3. Les systèmes distribués à grande échelle

3.1. Introduction [1.6]

Les termes *système distribué à grande échelle* (SDGE) ou *système de calcul global* sont utilisés pour désigner les environnements de développement et d'exécution pour les applications sur Internet. On peut considérer les SDGE comme une extension du concept de vol de cycles à l'échelle d'Internet i.e. des machines volontaires qui se connectent à un serveur pour recevoir des tâches à exécuter. De tels systèmes se basent sur le 'Pull model' c'est-à-dire qu'ils demandent périodiquement au serveur, du travail puisque les ressources participantes utilisent généralement des 'Firewalls'. Le système volontaire n'est pas symétrique car ce sont les (nœuds) volontaires qui fournissent les ressources informatiques aux projets, et non le contraire**.**

3.2. Définitions [1.7]

Le terme système réparti à grande échelle est apparu dans les années 1990 dans le cadre des systèmes répartis à grande échelle, le terme d'échelle est utilisé dans un sens quantitatif, c'est-à-dire qu'un système grande échelle est fortement réparti en terme de nombre de tiers impliqués, d'éloignement géographique entre les tiers, de temps de communication entre les tiers. De même, le terme « passage à l'échelle » est utilisé pour désigner la capacité d'un système à passer d'une répartition faible (peu de tiers, tiers proches, réseau local) à une répartition plus large (beaucoup de tiers, tiers très éloignés géographiquement, temps de communication important entre les tiers).

3.3. Caractéristiques fondamentales des systèmes distribués à grande échelle [1.8]

- ✓ *Extensibilité :* jusqu'à 100 k voire 1 M machines
- ✓ *Hétérogénéité :* différents matériels et OS
- ✓ *Dynamicité :* nombre de clients et de serveurs évoluent constamment
- ✓ *Disponibilité :* le propriétaire d'une ressource doit pouvoir définir une politique de partage de sa ressource
- ✓ *Tolérance aux pannes :* le système (et peut être les applications) doivent supporter l'existence d'éléments défaillants
- ✓ *Utilisabilité :* malgré les propriétés précédentes, le système doit rester facilement programmable et maintenable
- ✓ *Sécurité :* le système doit être sécurisé pour les participants, les serveurs et l'application. Un comportement malicieux ne doit pas pouvoir corrompre l'application. Un agent externe ne doit pas pouvoir se faire passer pour un serveur.

3.4. Classifications des systèmes distribués à grande échelle [1.9]

en générale, les technologies de SDGE cherchent à répondre à la problématique suivante : mieux exploiter les ressources existantes pour fournir de plus grande moyens de calcul et de stockage en toute transparence, concrètement, il s'agit d'autoriser un ou plusieurs utilisateurs, d'utiliser à moindre cout des ressources dont ils n'ont pas l'accès individuellement. Le lien entre les ressources et les utilisateurs s'opère par une couche intergicielle (middleware).

Elle est la brique de base représentant l'ensemble des éléments pour la mise en œuvre d'un SDGE.

Pour la réalisation de ces objectifs, nous distinguons deux inspirations ou courants majeurs pour les SDGE : Institutionnel ou Professionnel et Communautaire, comme illustré sur la figure suivante :

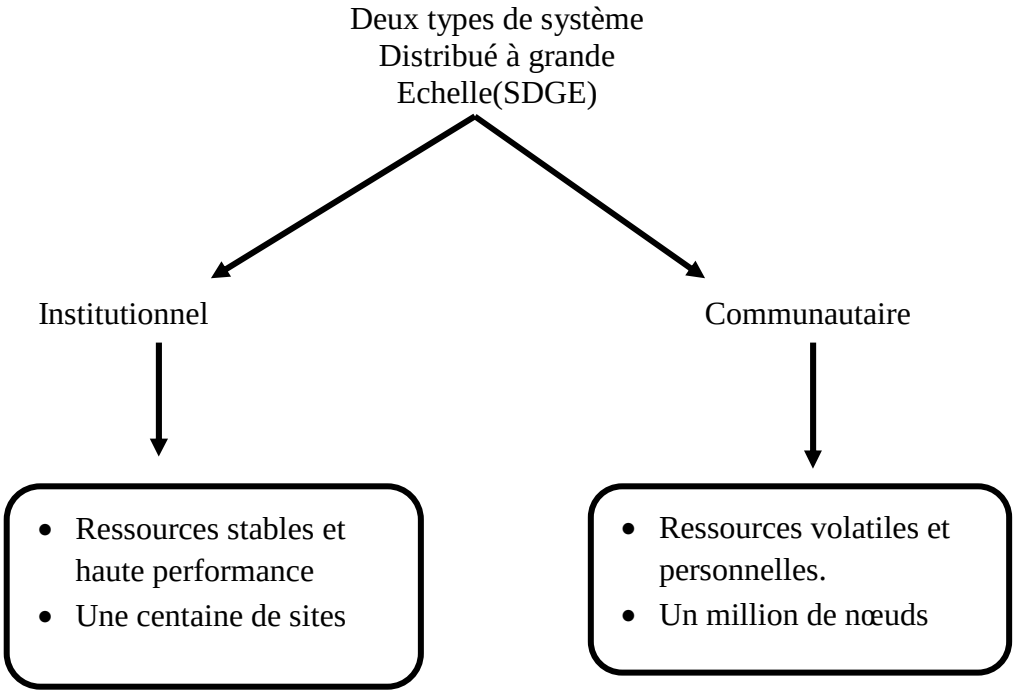


Figure 1.2 : Taxinomie des systèmes distribués à grande échelle [1.9].

4. Etat de l'art sur les grilles de calcul

4.1. Historique

Le mot « grille » vient de l'anglais *grid* qui a été choisi par analogie avec le système distribution d'électricité américain (*electric power grid*), ce terme été répandu en 1998 par l'ouvrage de Ian Foster et Carl Kesselman [**1.10].** En effet, une grille peut être vue comme un instrument qui fournit de la puissance de calcul et/ou de la capacité de stockage de la même manière que le réseau électrique fournit de la puissance électrique. La vision des inventeurs de ce terme est qu'il sera possible, à terme, de se « brancher » sur une grille informatique pour obtenir de la puissance de calcul et/ou de stockage de données sans savoir ni où ni comment cette puissance est fournie, à l'image de ce qui se passe pour l'électricité.

4.2 Définition [1.10]

Nous commençons par définir ce qu'est un site d'une grille.

4.2.1. Définition d'un site [1.10]

Un site est un ensemble de *ressources informatiques localisées géographiquement dans une même organisation* (campus universitaire, centre de calcul, entreprise ou chez un individu) et qui forment un domaine d'administration autonome, uniforme et coordonné.

Les *ressources informatiques* sont aussi bien des liens réseau (câbles, routeurs ou *switchs*) des machines (simples PC ou calculateurs parallèles) ou des éléments logiciels. Nous pouvons maintenant définir une *grille informatique*.

4.2.2 Définition d'une grille [1.10]

Une grille est un ensemble de ressources géographiquement éloignées sur plusieurs sites, partagées, distribuées, hétérogènes, délocalisées et autonomes et étant organisées virtuellement (les relations entre les entités qui la composent n'existent pas sur le plan matériel mais d'un point de vue logique), afin de réaliser des calculs impossibles à effectue sur un seul site.

Il est important de noter que les grilles informatiques sont, par nature, dynamiques :

1) les sites peuvent quitter ou rejoindre la grille à tout moment ;

2) de même, au sein de chaque site, de nouvelles ressources peuvent être ajoutées, d'autres peuvent tomber en panne ou être déconnectées.

La **figure1.4** représente un exemple de grille de calcul :

Figure 1.3: Grille de calcul [1.10].

4.3. Types des grilles [1.11]

Les grilles peuvent être partagées en quatre types selon les traitements qui s'exécutent sur leurs nœuds:

4.3.1. Grille d'information (Information Grid)

La ressource partagée est la connaissance. L'Internet est le meilleur exemple : un grand nombre de machines hétérogènes réparties sur toute la surface du globe autorisant un accès transparent à l'information.

4.3.2. Grille de stockage (Desktop Grid)

L'objectif de ces grilles est de mettre à disposition un grand nombre de ressources de stockage d'information afin de réaliser l'équivalent d'un "super disque dur" de plusieurs PetaBytes. Le projet DataGrid est un bon exemple de grille de stockage.

4.3.3. Gille de serveur (Server Grid)

C'est une grille dont les ressources de calcul sont plus particulièrement mises en avant par rapport au réseau ou aux ressources de stockage.

Ce type de grille est intéressant pour les applications gourmandes en temps de calcul et qui ne nécessitent pas ou peu de communications en termes de quantité de données à échanger. On classe généralement ce type d'applications sous l'appellation Calcul Hautes Performances (HPC : High Performance Computing). Dans cette approche, l'utilisateur contacte et demande aux ressources, selon l'avis de l'ordonnanceur, d'effectuer du travail (par exemple une opération algébrique ou la résolution d'un problème donné)**.**

4.3.4. Grille globale (Global Grid)

Pour désigner une grille dont l'ensemble des ressources constitue un supercalculateur virtuel sur lequel l'utilisateur soumet ses applications utilisant des bibliothèques de fonctions spécifiquement conçues.

4.4. Différentes topologies de grilles [1.12]

On peut énumérer les grilles d'un point de vue topologique en trois types par ordre croissant d'étendue géographique et de complexité: intra grilles (intragrids), extragrilles (extragrids) et intergrilles (intergrids).Qu'on détaillera si après.

4.4.1. Intragrille (en analogie avec Intranet)

Cette topologie est composée d'un ensemble relativement limité 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 à haut débit, un domaine de sécurité unique et maîtrisé par les administrateurs de l'organisation et un ensemble relativement statique et homogène de ressources.

4.4.2. Extragrille (en analogie avec Extranet)

Une extragrille étend le modèle en regroupant plusieurs intragrilles. Les principales caractéristiques d'une telle grille sont la présence d'un réseau d'interconnexion hétérogène haut et bas débit, 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.

4.4.3. Intergrille (en analogie avec Internet)

Une intergrille 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, de plusieurs domaines de sécurité distincte et ayant parfois des politiques de sécurité différente et même contradictoire, et d'un ensemble très dynamique de ressources.

4.5. Architecture d'une grille

L'architecture d'une grille est organisée en couches. 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 **[1.13].**

La **figure1.4** représente les couches de la grille par analogie aux couches d'Internet.

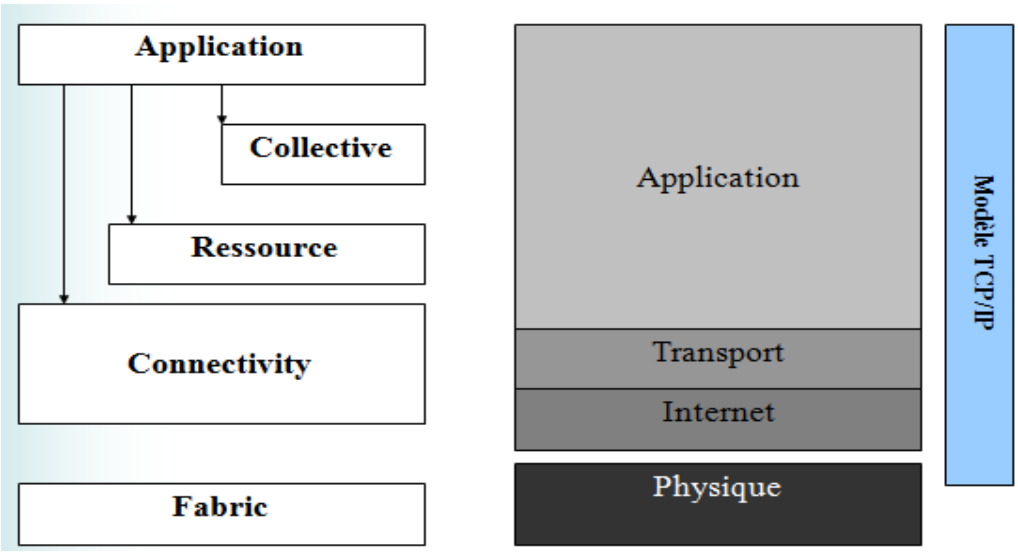


Figure 1.4 : les différentes couches de la grille par analogie aux couches du protocole Internet [1.14].

La couche Fabrique [1.13]

La *couche Fabrique* fournit les ressources. Ce sont d'un point de vue physique des ressources telles que des processeurs pour le calcul, du stockage, des bases de données, des annuaires ou des ressources réseau. Une ressource peut-être également une entité logique comme un système de fichiers distribué, ou un serveur virtuel dans le cas d'un *cluster* d'ordinateurs. La couche Fabrique est la plus basse du modèle. Elle est en relation directe avec le matériel pour mettre à disposition les ressources partagées.

Lorsqu'une demande d'accès à une ressource est formulée, par le biais d'une opération de partage d'un niveau supérieur, un composant logiciel du niveau Fabrique est invoqué. Le rôle de ce composant est d'agir directement sur les ressources logiques ou physiques de la grille.

La couche Connectivité [1.13]

La *couche Connectivité* implémente les principaux protocoles de communication et d'authentification nécessaire aux transactions sur un réseau de type grille. Les protocoles de communication permettent l'échange des données à travers les ressources du *niveau Fabrique.* Ces protocoles d'authentification s'appuient sur les services de communication pour fournir des mécanismes sécurisés de vérification de l'identité des utilisateurs et des ressources.

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.

La couche Ressource [1.13]

Utilise les services des *couches Connectivité* et *Fabrique* pour collecter des informations sur les caractéristiques des ressources, les surveiller et les contrôler. Elle s'occupe également de l'aspect facturation. La *couche Ressource* ne se préoccupe pas des ressources d'un point de vue global, elle ne s'intéresse pas à leur interaction. Ceci incombe à la *couche Collectif* abordée dans la section suivante. La *couche Ressource* ne s'intéresse qu'aux caractéristiques intrinsèques des ressources et à la façon dont elles se comportent.

La couche Collectif [1.13]

La *couche Collectif* se charge des interactions entre les ressources. Elle gère l'ordonnancement et la co-allocation des ressources en cas de demande des utilisateurs faisant appel à plusieurs ressources simultanément. C'est elle qui choisit sur quelle ressource de calcul faire exécuter un traitement en fonction de coûts qu'elle sait estimer. Elle s'occupe également des services de réplication des données. En outre, elle a en charge la surveillance des services et elle doit assumer la détection des pannes. En un mot elle a un rôle « d'orchestrateur » de l'ensemble des ressources du système.

La couche Application [1.13]

La couche la plus haute du modèle est la *couche Application* qui correspond aux logiciels qui utilisent la grille pour fournir aux utilisateurs ce dont ils ont besoin, qu'il s'agisse de calcul, ou de données. Les applications utilisent des services de chacune des couches de l'architecture. Les *couches Collectif et Ressources* sont sollicitées pour la recherche des ressources. Après les avoir trouvées, et après s'être authentifiées au travers de la *couche Connectivité*, les applications utilisent les services du *niveau Fabrique* pour y accéder.

4.6. Caractéristiques d'une grille de calcul [1.15]

1) Une grille de calcul est partagée par plusieurs utilisateurs indépendants. Ceci conduit à une grande variabilité dans l'utilisation des ressources mises à disposition par les grilles.

2) La grille est repartie sur différents sites qui ne sont pas sous administration commune. Cela conduit à une grande hétérogénéité tant au niveau matériel que de l'environnement logiciel.

3) chaque site d'une grille peut utiliser ces propres machines, avec son propre système d'exploitation. Les politiques de sécurité peuvent aussi être différentes d'un site à l'autre.

4) La grille n'a pas une structure statique. Que ce soit du fait de pannes matérielles, de remplacements ou d'ajouts, des ressources peuvent apparaitre ou disparaitre à tout instant.

5) L'échelle des grilles de calcul est grande. Le nombre de processeurs d'une grille de calcul se compte par milliers.

4.7 Quelque avantages et inconvénients d'une grille de calcul [1.15]

4.7.1 Avantages

Exploiter les ressources sous utilisées : Les études montrent que les ordinateurs personnels et les stations de travail restent inactifs la plupart du temps. Les grilles de calcul permettent ainsi d'utiliser les cycles processeurs durant lesquels les machines sont inactives afin d'exécuter une application ou une partie d'application nécessitant une puissance de calcul importante.

Fournir un cadre distribué : Une grille de calcul peut agréger une importante quantité de ressources afin de fournir une puissance de calcul aussi performante que les gros calculateurs parallèles. Les ressources agrégées peuvent aller du simple PC à des calculateurs parallèles.

Fiabilité et disponibilité des services : Les ressources fédérées par une grille de calcul sont géographiquement éloignées et disponibles en importantes quantités. Cela permet d'assurer la continuité du service si certaines ressources deviennent indisponibles. Dans ce cas, les logiciels de contrôle et de gestion de la grille sont en mesure d'allouer d'autres ressources et de leur transmettre les calculs à effectuer.

Gestion adéquate des ressources : En partageant les ressources, une grille peut fournir l'accès à des ressources spéciales comme des équipements (bras robotiques, cameras, ...) ou des bibliothèques (BLAS, LAPACK). Ainsi ces ressources seront utilisées et partagées par plusieurs utilisateurs.

4.7.2. Inconvénients

1) L'architecture de la grille est caractérisée par une grande hétérogénéité des ressources et par l'absence d'une connaissance à priori des ressources mises à disposition.

2) Le grand nombre de ressources et leur partage rend difficile leur gestion.

4.8. Les services fournis par les systèmes de grille de calcul [1.6]

Les systèmes de grilles de calcul peuvent être employés pour fournir les services suivants :

4.8.1. Services de calcul (Computational Services)

Il s'agit de fournir des services sécurisés pour exécuter les tâches des applications sur les ressources informatiques distribuées. Quelques exemples de grilles informatiques sont NASA IPG, World-Wide Grid, et NSF TeraGrid.

4.8.2. Services de données (Data Services)

Ils cherchent à permettre le transfert sécurisé aux ensembles de données distribuées et à leur gestion. Un exemple d'application qui a besoin d'un tel service pour la gestion, le partage et le traitement de grands ensembles de données est la Physique des Hautes Energies.

4.8.3. Services d'application (Application Services)

Ceux-ci ont pour objectif de gérer des applications et de fournir un accès à distance aux logiciels et aux bibliothèques d'une manière transparente. Un exemple qui peut être employé pour développer de tels services est NetSolve.

4.8.4. Services d'information (Information Services)

Ceux-ci se spécialisent dans l'extraction et la présentation des données significatives en employant les services données et applications.

4.8.5. Services de connaissance (Knowledge Services)

Ils se focalisent sur la manière d'acquérir, d'employer, de rechercher, d'éditer, et de maintenir des connaissances pour aider les utilisateurs à atteindre leurs buts et objectifs particuliers.

4.9. Les intergiciels des grilles de calculs (middleware)

Pour construire une grille, on utilise un ensemble de logiciels appelé *intergiciel*. L'intergiciel unifie l'accès à des ressources informatiques hétérogènes. Il se place entre les systèmes d'exploitation existants (pas de perturbation de ces systèmes) et l'utilisateur.

Il masque à ce dernier l'hétérogénéité des divers systèmes installés et fournit un ensemble de routines (commandes et bibliothèque de programmation) indépendantes du système. Ces

commandes et fonctions permettent par exemple d'exécuter une application sur une machine avec un langage de description de ressources (quantité de mémoire nécessaire, temps CPU) indépendant de la machine utilisée **[1.16].**

Le middleware utilisé par un grand nombre de projets de grille, GT (Globus Toolkit), la grille scientifique américaines OGS (Open Grid Science) et la grille européenne EGEE l'utilisent depuis leur création mais EGEE utilise maintenant une évolution de middleware appelé GLite

Dans cette section, nous présentons Globus Toolkit et Glite **[1.15].**

4.9.1. Globus Toolkit [1.15]

Pratiquement la plupart des projets mondiaux de grille de calcul reposent sur un ensemble de programmes : le *Globus Toolkit*. Ce *toolkit* est développé par la *Globus Alliance*.Cette organisation est constituée principalement par l'´equipe d'Ian Foster au laboratoire national d'Argonne et par l'´equipe de Carl Kesselman à l'université de Caroline du Sud à Los Angeles. Il s'agit d'un intergiciel, c'est à dire d'un ensemble de briques de base.
Ces différents modules fournissent tous les services nécessaires à l'´elaboration d'une grille comme la sécurité, la localisation de ressources et la gestion de celles-ci.

Le *Globus Toolkit* inclut :

1. GRAM (*Globus Ressource Allocation Manager*), qui a pour rôle de convertir des requêtes de la grille en commandes compréhensibles par les différentes ressources.
2. GSI (*Grid Security Infrastructure*) dont le rôle est la sécurité et l'authentification des utilisateurs.
3. MDS (*Monitoring and Discovery Service*) dont le rôle est de collecter et de stocker des informations sur les différentes ressources (capacité de stockage, puissance de calcul, bande passante entre nœuds,...).
4. GRIS (*Grid Ressource Information Service*) qui maintient une liste de l'´etat des ressources, de leur configuration et de leur capacité du moment.
5. GIIS (*Grid Index Information Service*) qui coordonne les différents services GRIS.
6. GridFTP dont le rôle est de fournir un service de transfert robuste, fiable et rapide.

7. Le *Replica Catalog*, dont le rôle est de maintenir une liste des différents emplacements dans lesquels sont stockés les fichiers ainsi que leurs copies. Un fichier peut en effet être répliqué à plusieurs endroits de la grille, soit de manière automatique, soit par l'utilisateur.
8. Le *Replica Management System* dont le rôle est de permettre aux applications de créer des copies de certaines données et de gérer celles-ci.

Les modules sont classés selon 4 types : sécurité, gestion des données, gestion des jobs, monitoring des services et des ressources. La majorité de ces services utilisent HTTP pour les transferts ce qui facilite le déploiement par Internet.

Le succès du *Globus Toolkit* provient de deux raisons principales :

- Le *Globus Toolkit* est orienté-objet. On peut ainsi y sélectionner exactement les services désirés. Ceci permet une intégration beaucoup plus en douceur des logiciels qui doivent être adaptés à la grille. Une application peut ainsi utiliser par exemple uniquement le GRIS ou le GRAM sans utiliser les autres services.
- Le *Globus Toolkit* est un ensemble de logiciels libres.

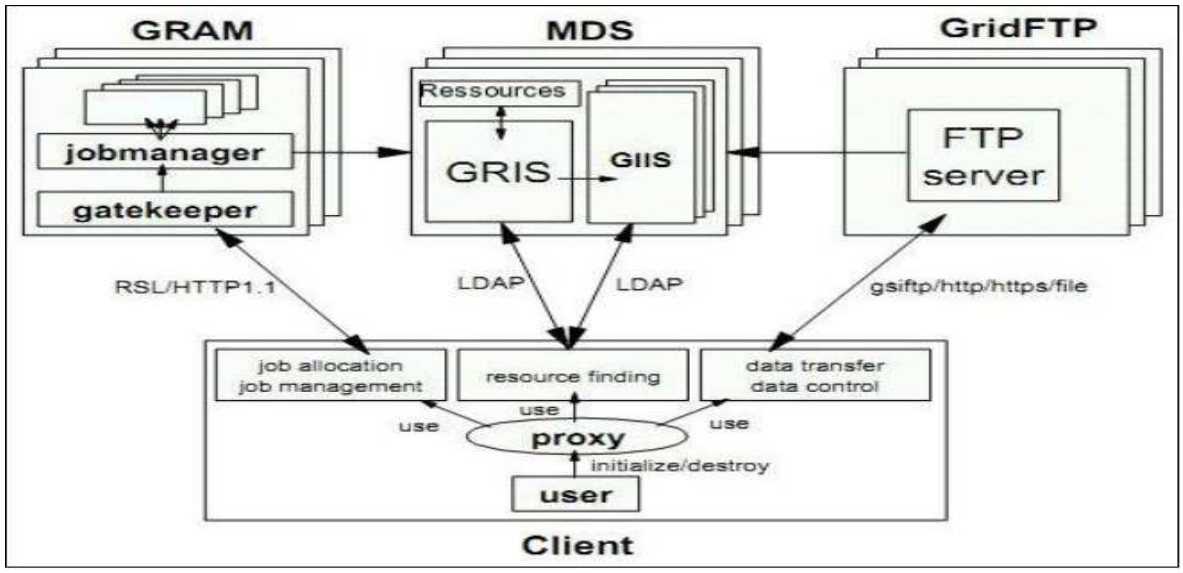


Figure 1.5: Vue conceptuelle des différents composants de GT [1.17].

4.9.2. GLite [1.18]

L'intergiciel gLite (ou Lightweight Middleware for Grid Computing) est le fruit de nombreux autres projets de grilles de calcul, il intègre des composants provenant des différents projets d'intergiciels en cours, tels *Condor*, le *Globus Toolkit*. Cet intergiciel a été développé par EGEE (Enabling Grid for E-sciencE) et WLOG (Worldwide LHC Computing Grid Project).

GLite est une nouvelle génération de middleware distribué sous une licence de type logiciel libre et utilisé pour la gestion et l'administration dune grille de calcul dite légère.
GLite est basé sur une architecture Service Oriented Architecture (SOA), SOA est un paradigme architectural par le biais du quel on peut concevoir des infrasctures permettant aux clients et aux fournisseurs d'échangés des services. Ce type d'architecture permet de garantir une meilleure conformité aux prochaines normes et standard grid tel que le web.

4.10. Exemples des normes de grille [1.19]

Dans la section précédente, nous avons discuté les technologies requises dans l'implémentation de grille.

Dans cette section nous regardons certains des standards ouverts utilisés pour la mise en œuvre d'une grille.

4.10.1. De la grille au Web Service

Les services de la grille ainsi définie par OGSA (Open Grid Service Architecture) sont une prolongation des Web Services. Ainsi, les services de grille peuvent accroître les caractéristiques de disponibilités des Web Services.
Nous discutons les normes les plus fondamentales du Web Service à savoir.

4.10.1.1. Le XML (eXtensible Markup Language)

XML est un langage de balisage dont le but est de faciliter le partage des données à travers des interfaces utilisant un format commun. Il forme la base des services Web. Tous les messages échangés dans des Web Services adhérents au format de document de XML.

4.10.1.2. Protocole d'accès simple aux objets (SOAP)

C'est un protocole basé sur la transmission de messages, qui peut être employé pour la communication via Internet de deux nœuds ou sites. Les messages du protocole SOAP sont basés sur XML et sont par conséquent indépendants de plateforme. Elle forme la base du Web Services. Des messages de SOAP sont transmis via le HTTP.

Ces messages sont différents de l'autre technologie telle le RPC ou CORBA, les messages de SOAP conviennent quand les messages sont de petite taille de plus en plus la taille du message augmente, son entête augment également ce qui par conséquent réduit l'efficacité de la communication.

4.10.1.3. Langue de définition de service de Web (WSDL)

WSDL est un Document XML employé pour décrire l'interface du Web service. Un document WSDL décrit un Web Service en utilisant les éléments principaux suivants :

- ✓ **port Type**: L'ensemble d'opérations effectuées par le web service chaque Opération est définie par un ensemble de messages d'entrée et de rendement.
- ✓ **Message**: Il représente les messages employés par le service de Web. C'est une Abstraction des données étant transmises.
- ✓ **Type**: Il se rapporte aux types de données définis pour décrire le message échangé.
- ✓ **Attache :** Elle spécifie le protocole de transmission employé par le Web Service.
- ✓ **Port**: Il définit l'adresse obligatoire pour le Web Service.
- ✓ **Service**: Il est employé pour agréger un ensemble de ports relatifs et les regroupe pour pouvoir fournir Web Services.

4.10.1.4. Description, découverte et intégration universelles (UDDI)

C'est un annuaire à base de XML utilisé pour trouver un Web Service sur internet. C'est des spécifications qui permettent à des entreprises et e-commerces d'éditer leurs informations et leurs Web Services permettant à d'autres Web Services de localiser ces informations.

Un enregistrement d'UDDI est une liste de service basée sur XML. Chaque liste contient l'information nécessaire exigée pour pouvoir retrouver et se lier à des Web Services particuliers.

4.10.2. Open Grid Service Architecture (OGSA)

(OGSA) définis un cadre de service de base pour le Web Service pour la mise en œuvre d'une grille. Elle cherche à normaliser les services fournis par une grille telle que la découverte de ressource, gestion des ressources, sécurité, etc. Par une interface de Web Service standard. Elle définit également ces dispositifs qui ne sont pas nécessairement utiles pour l'implémentation d'une grille. OGSA est basé sur des caractéristiques existantes de Web service et ajoute des dispositifs aux Web Service pour la rendre appropriée à l'environnement de la grille.

4.10.3. Open Grid Services Infrastructure (OGSI)

OGSA décrit les dispositifs qui sont nécessaires pour l'exécution des services fournis par la grille, comme du Web Service. Cependant, ne fournis pas les détails de l'exécution. OGSI fournit des spécifications formelles et techniques requises pour l'exécution des services de la grille. Elle fournit le langage de description de web service (WSDL), qui définit un service de grille. OGSI fournit également les mécanismes pour la création, la gestion et l'interaction parmi ces différents services.

4.10.4. Web Service Resource Framework (WSRF)

La motivation derrière le développement du WSRF est de définir un cadre générique et ouvert pour modelé et accéder aux ressources totales en utilisant des Web Services. Il définit des conventions pour permettre la gestion de l'état des applications à découvrir et éventuellement agir l'un sur l'autre avec les autres Web Service.

Les Web Service standards n'ont pas une notion d'état de la grille, les applications ont besoin de la notion d'état parce qu'ils exécutent souvent une série de demandes là où le produit d'une opération peut dépendre du résultat des opérations précédentes. Le WSRF peut être employé pour développer une telle grille complète de services. Le format de l'échange de message dans WSRF est défini par le WSDL.

WSRF est soutenu par de diverses compagnies et les spécifications ont été finalisées par le comité de fonctionnement d'OASIS.

4.10.5. OGSA-DAI

Open Grid Service Architecture-Data Access and Intégration (OGSA-DAI) est un projet conçu par le groupe de travail BRITANNIQUE de base de données. Le but de ce projet est de développer l'intergiciel pour fournir l'accès et l'intégration à distribuer points d'émission en utilisant une grille.

Cet intergiciel fournit l'appui pour différentes données sources telles que les bases de données relationnelles et de XML. Ces points d'émission peuvent être questionnés, mis à jour et transformés par l'intermédiaire du Web service d'OGSA-DAI.

Ces services Web peuvent être déployés dans une grille, de ce fait faisant la grille de points d'émission permise. La demande au service de Web d'OGSA-DAI d'accéder à un point d'émission est indépendante du point d'émission servi par le service de Web. Les services de Web d'OGSA sont conformes avec l'interopérabilité du Web Service (WS-I) et le WSRF

4.11. Exemples des projets de grilles

4.11.1. Grid5000: la grille expérimentale française [1.20]

Un important projet de grille de calcul a été lancé en 2003 : le projet Grid'5000. Son but est de fédérer des grappes de calculateurs réparties dans 9 différents sites géographiques en France : Bordeaux, Grenoble, Lille, Lyon, Nancy, Orsay, Rennes, Sophia-Antipolis et Toulouse. **Figure 1.6** représente les 9 sites interconnectés par le réseau *Renater* (correspondant aux traits reliant les sites sur la carte de la Figure 1.6). Ce réseau est en cours de transition et va offrir un débit de 10 Gb/s pour des latences allant de 3 à 14 ms3. À terme, Grid'5000 devrait comporter 5000 processeurs répartis dans ces 9 sites. Au sein des sites, les nœuds sont interconnectés par des réseaux à très faible latence comme *Myrinet* (pour lequel la latence entre deux nœuds est de l'ordre de 2 us).). Il y a donc un rapport de latence de l'ordre d'un facteur 10.000 entre les liens intergrappe et les liens intragrappe.

La topologie de cette grille est présentée dans la figure suivante :

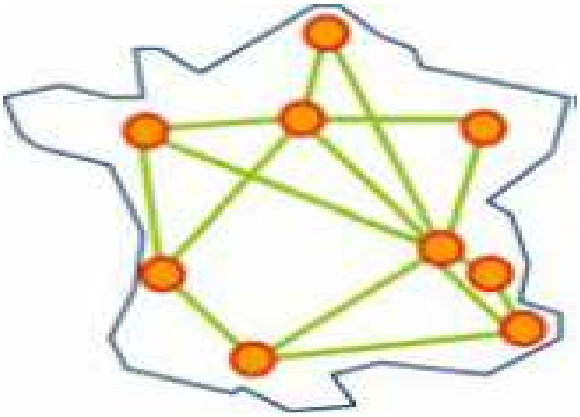


Figure 1.6 : Topologie de la grille grid5000 [1.20].

4.11.2 EGEE [1.21]

(Enabling Grid for E-sciencE) Débuté initialement le 1er avril 2004. Elle a était poursuivie d'une deuxième phase de deux ans a commencée du 1er avril 2006, et est financée par la Commission européen. L'objectif d'EGEE est de construire sur avances récentes dans la technologie des grilles, pour fournir une infrastructure de production, basée sur des logiciels de production de qualité, qui seront à la disposition des scientifiques 24 heures sur 24.
EGEE fournis la recherche académique et l'industrielle un accès aux ressources informatiques importantes et fiables, indépendamment de leurs situations géographiques.
EGEE est axée sur deux domaines de recherche qui sont les suivants:

- ✓ **La physique de haute énergie**: la grille de calcul destiner à gérer les petabytes générer par le large collisionneur de particules du CERN (centre européen de recherche nucléaire) annuellement
- ✓ **Bioinformatique**: la grille biomédicale aide des communautés de chercheures en médecine et biologie à effectuer des calculs volumineux et réaliser des traitements sur des images médicales.

La grille EGEE est l'un des projets européens les plus couteux, avec un budget dépassant les tarentes millions d'euros pour les premières années 2004-2005 avec l'intervention de plus de 71 institutions reparties sur plus de 11 grilles régionales, actuellement elle comporte plus de 8000 processeurs distribués sur 50 sites gérant une capacité de stockage ~5 petabytes

La **figure 1.7** donne un aperçu sur les pays intervenant dans EGEE.

Cette grille est bâtie sur le réseau européen à haut débit GEANT, et vient pour succéder au projet DataGrid.

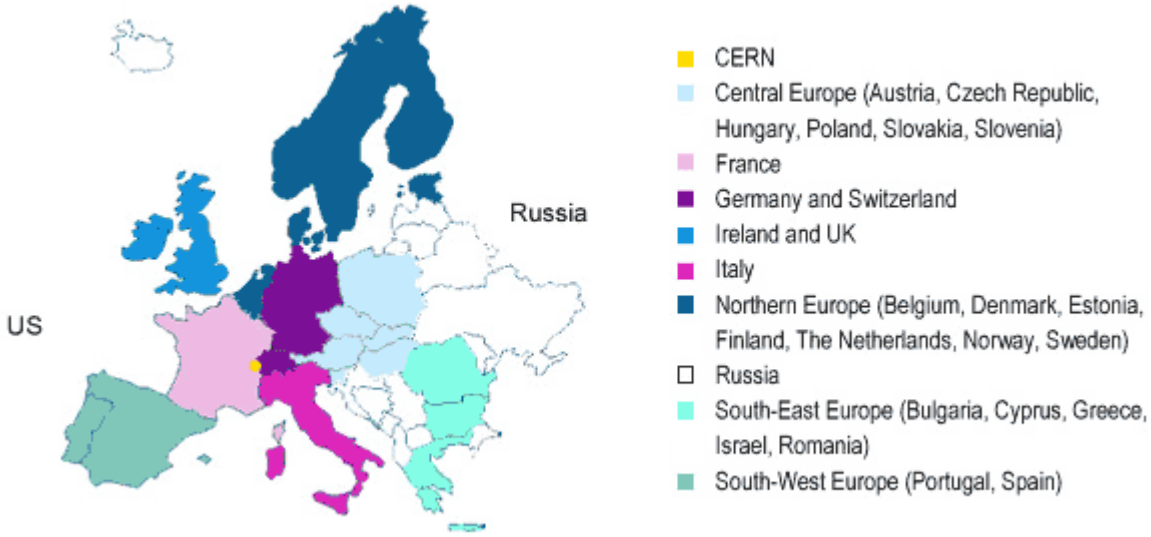


Figure 1. 7: les pays intervenant dans EGEE [1.21].

5. Cloud

5.1. Introduction [1.22]

Le Cloud computing est une nouvelle manière de fournir et d'utiliser les aptitudes des systèmes informatique basé sur un *nuage* (*cloud* en anglais), c'est-à-dire un parc de machines, d'équipement de réseau et de logiciels. Le nuage est sous la responsabilité du fournisseur qui en dissimule les détails techniques, et le consommateur peut disposer des aptitudes du nuage qui sont mises en self-service, disponibles via Internet et payées à l'usage Les grandes entreprises

du secteur informatique telles que IBM, Microsoft, Google, Dell et Sun soutiennent massivement le cloud computing qui est parfois considéré comme un important changement des systèmes informatiques et fait l'objet d'un certain battage médiatique.

5.2. Historique [1.23]

Il n'y a pas de date-clé à laquelle nous puissions dire que le Cloud Computing est né. La notion de Cloud fait référence à un nuage, tel que l'on a l'habitude de l'utiliser dans des schémas techniques lorsque l'on veut représenter Internet. Un réseau comme Internet est constitué d'une multitude de systèmes fournissant des services et des informations. Le Cloud Computing est dans cette lignée : un ensemble de services et de données consommables.

Cette notion de consommation a été proposée en 1961, lors d'une conférence au MIT (Massachusetts Institute of Technology), par John McCarthy **[1.23]** aussi connu comme l'un des pionniers de l'intelligence artificielle (dont il proposa le nom en 1955) et pour avoir inventé LISP en 1958.

Lors de ce discours, John McCarthy suggéra que la technologie informatique partagée (« time-sharing ») pouvait construire un bel avenir dans lequel la puissance de calcul et même les applications spécifiques pouvaient être vendues comme un service public.

C'est donc depuis presque 50 ans que l'idée d'une informatique à la demande est présente dans les esprits même si les technologies n'étaient jusqu'alors pas au rendez-vous pour pouvoir concrétiser cette idée.

Avec les différents progrès technologiques réalisés durant ces 50 dernières années, tant sur le plan matériel, logiciel et conceptuel, aux avancées des mécanismes de sécurité, à l'élaboration de réseaux complexes mais standardisés comme Internet, et à l'expérience dans l'édition et la gestion de logiciels, services, infrastructures et stockage de données, il est maintenant possible d'entrer dans l'ère du Cloud Computing, telle que rêvait par John McCarthy en 1961.

5.3. Définition

Le Cloud Computing, littéralement l'informatique dans les nuages est un concept qui consiste à déporter sur des serveurs distants des stockages et des traitements informatiques traditionnellement localisés sur des serveurs locaux ou sur le poste de l'utilisateur. Il consiste à proposer des services informatiques sous forme de service à la demande, accessible de n'importe où, n'importe quand et par n'importe qui, L'idée principale à retenir est que le Cloud n'est pas un ensemble de technologies, mais un modèle de fourniture, de gestion et de consommation de services et de ressources informatiques **[1.24].**

Les applications et les données ne se trouvent plus sur l'ordinateur local, mais «Dans le Cloud » composé d'un certain nombre de serveurs distants, interconnectés au moyen d'une excellente bande passante. L'accès au service se fait par une application standard facilement disponible, la plupart du temps un navigateur web **[1.25].**

Actuellement, l'utilisation du Cloud Computing s'est démocratisé avec l'accès à des connexions internet dites « haut débit » et à la vulgarisation des ordinateurs de grande capacité, et ayant une grande puissance de traitement. Il peut être considéré comme la cinquième évolution de l'informatique après le mainframe, le PC, le client serveur, le web et pour finir le Cloud Computing d'après Microsoft. **[1.25]**

Le Cloud Computing peut être comparé à la distribution de l'énergie électrique. Bien que chaque particulier pourrait produire sa propre énergie électrique (solaire, éolienne), un opérateur le fait à grande échelle et propose ce service au client à travers un réseau de distribution (câble électrique). Chaque client est facturé uniquement en fonction de sa consommation. On retrouve dans cet exemple des analogies dans la terminologie du Cloud Computing et de la distribution de l'énergie électrique **[1.26].**

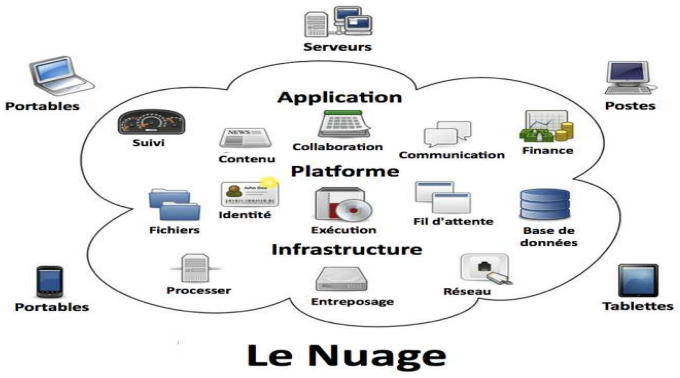


Figure 1.8 : Illustration d'un Cloud Computing [1.26].

Le Cloud Computing est une combinaison de plusieurs concepts :

- **Entrepôt de données** (data centre) : un bâtiment héberge les serveurs physiques et fournit l'ensemble des besoins en ressources informatiques,
- **Virtualisation** : pour obtenir de la puissance supplémentaire, les entreprises utilisent des serveurs virtuels, hébergés sur les serveurs physiques du centre de calcul. Le recours à cette virtualisation « à la carte », et la flexibilité qu'elle offre, est l'un des atouts clés du Cloud Computing.

5.4. Architecture de Cloud computing [1.27]

Le Cloud Computing propose trois modèles principaux comme le montre la **figure 1.9**.

- L'infrastructure (Iaas : Infrastructure as a service)
- La plate-forme (Pass : Platform as a service)
- L'application (Saas : Software as a service)

La figure suivante représente l'Architecture d'un Cloud Computing.

Figure 1.9: Architecture d'un could computing [1.27]

5.4.1. Iaas: Infrastructure as a service

L'infrastructure en tant que service ou, en anglais, Infrastructure as a service (Iaas) est le premier modèle de cloud on désigne une infrastructure matérielle, louée à la demande : stockage, machines virtuelles, OS, etc… ou :

L'entreprise maintient : les applications, les runtimes, l'intégration SOA, les bases de données, les logiciels serveurs.

Le fournisseur Cloud maintient : la virtualisation, le matériel du serveur, le stockage, les réseaux.

Principaux acteurs : IBM, Amazon, SUN (maintenant ORACLE), VMWARE, GO GRID, DELL, etc.

5.4.2. Paas: Platform as a service

La plate-forme en tant qu'un service désigne la mise à disposition d'un environnement de développement et d'exploitation de logiciels sur Internet. Ces plates-formes sont donc directement utilisables par des éditeurs qui proposeront leurs logiciels en mode Cloud. Ou l'entreprise maintient uniquement les applications.

Le fournisseur Cloud maintient : les runtimes, l'intégration SOA, les bases de données, le logiciel serveur, la virtualisation, le matériel serveur, le stockage, les réseaux.

Principaux acteurs: Google (GAE), force.com, Microsoft (Azure), SpringSource (CloudFoundery), etc.

5.4.3. SaaS: Software as a service

Le logiciel en tant qu'un service désigne la mise à disposition par Internet d'applications informatiques (logiciels) comme un service dans le cadre d'un abonnement, C'est en quelque sorte la partie visible du Cloud Computing pour l'utilisateur final, qui n'a plus besoin d'installer l'application sur son poste, et qui accède à son compte par le Web,

Le fournisseur Cloud maintient : les applications, les runtimes, l'intégration SOA, les bases de données, le logiciel serveur, la virtualisation, le matériel serveur, le stockage, les réseaux.

Le *SaaS*, souvent associé au « cloud computing » peut être vu comme un modèle économique de consommation des applications : celles-ci sont consommées et payées à la demande (par utilisateur et par minute d'utilisation par exemple) et non plus acquises par l'achat de licences.

Principaux acteurs :

- Google (Apps)
- Zoho (Apps)
- saleforce.com, Microsoft, IBM, etc.

L'Infrastructure en tant que service offre une base matérielle (hardware) aux plateformes en tant que service. Ces infrastructures sont mises en place et gérées par des administrateurs réseau avec un bon niveau d'expertise. Elles sont le plus souvent constituées d'équipements réseaux et de serveurs la plus part du temps entièrement virtualisés.

La plateforme en tant que service est un ensemble de composants reposant sur l'infrastructure offerte par la couche *Iaas*. Elle permet aux développeurs d'applications d'avoir une plateforme de travail adaptable, distribuée et virtualisée dans laquelle ils n'ont plus besoin d'installer l'architecture sous-jacente (réseaux, matériels, serveur, système d'exploitation)

L'application en tant que service est une application souple et déployée dans une plateforme en tant que service. C'est une application souple qui est accessible uniquement à travers un réseau et qui est le plus souvent facturée à l'utilisateur final.

5.4.4. Modèles de service associés [1.27]

À côté de ces modèles de services, on trouve une multitude de modèles de service construits mécaniquement sur l'expression « as a service ». Voici par ordre alphabétique :

✓ **BPaaS :** il s'agit du concept de *Business Process as a service* (BPaaS) qui consiste à externaliser une procédure d'entreprise suffisamment industrialisée pour s'adresser directement aux managers d'une organisation, sans nécessiter l'aide de professionnels de l'informatique.

✓ **DaaS :** Le Desktop as a Service (aussi appelé en français « bureau en tant que service »,

« Bureau virtuel » ou « bureau virtuel hébergé ») est l'externalisation d'une Virtual Desktop Infrastructure auprès d'un fournisseur de services. Généralement, le *Desktop as a Service* est proposé avec un abonnement payant.

✓ **Network as a Service (NaaS):** Le Network as a Service correspond à la fourniture de services réseaux, suivant le concept de Software Defined Networking (SDN).

✓ **STaaS :** *STorage as a Service* correspond au stockage de fichiers chez des prestataires externes, qui les hébergent pour le compte de leurs clients. Des services grand public, tels que SugarSync et Box.net, proposent ce type de stockage, le plus souvent à des fins de sauvegarde ou de partage de fichiers. Voici d'autres exemples : Amazon Simple Storage Service, Dropbox, Google Drive, Ubuntu One.

5.5. Caractéristique d'un Cloud [1.28]

Libre service à la demande : Le consommateur récupère des ressources de calcul et de stockage à la demande (machine virtuelle).

Elasticité : Croissance ou décroissance dynamique du nombre de ressources de nuage en fonction de la demande et des besoins donnant une vue de ressources infinies, avec ce

dynamisme dés qu'une ressource se libère, on peut la louer à n'importe quel moment à un autre client.

Large accès au réseau : Des services sont mis tous au long du réseau et peuvent être accéder par des mécanismes standard qui favorisent l'utilisation et l'accès par de plateformes hétérogènes des clients (ex : les PC portable, mobile, PDA, etc.).

Une mise en commun des ressources: Les ressources de calcul sont mises à disposition des clients sur un modèle multi-locataires, avec une attribution dynamique des ressources physiques et virtuelles en fonction de la demande. Le client n'a généralement aucun contrôle ou connaissance sur l'emplacement exact des ressources fournies (ressources cachées). Toutefois, le client peut imposer de spécifier l'emplacement à un niveau plus haut d'abstraction (par exemple le pays, l'état ou le Data Center).

Service mesuré et facturation à l'usage : La facturation est calculée en fonction de la durée et de la quantité de ressources utilisées.

Résilience : Les services IBM de continuité d'activité et de *résilience* garantissent la continuité des opérations informatiques en continu.

5.6. Modèle de déploiements de Cloud computing [1.29]

Les nuages de calcul peuvent être classés en différentes modèle de déploiement. On observe les clouds publics, privés, communautaires, hybrides comme le montre la **figure 1.11**.

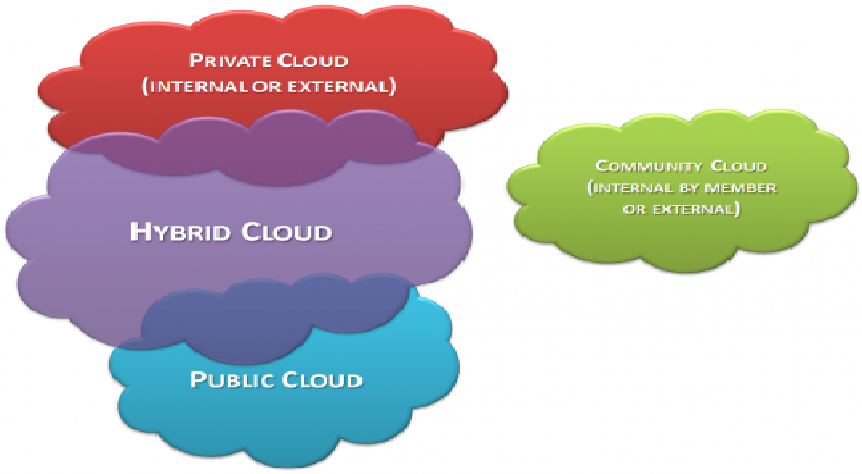


Figure 1.11 : Modèle de déploiements d'un Cloud [1.29].

5.6.1. Clouds publics

Ce type d'infrastructure est accessible à un large public et appartient à un fournisseur de « Cloud services ».

5.6.2. Clouds privés

L'infrastructure Cloud fonctionne pour une organisation unique. Elle peut être gérée par l'organisation elle-même (Cloud Privé interne) ou par un tiers (Cloud Privé externe). Dans ce dernier cas, l'infrastructure est entièrement dédiée à l'entreprise et accessible via réseaux sécurisés.

5.6.3. Clouds communautaires

L'infrastructure est partagée par plusieurs organisations qui ont des intérêts communs (par exemple les exigences de sécurité, de conformité …).
Comme le Cloud privé, il peut être géré par les organisations elles-mêmes ou par un tiers.

5.6.4. Clouds hybrides

L'infrastructure se compose de deux nuages ou plus (Privé, Communautaire ou Public), qui restent des entités uniques, mais qui sont liées par une technologie normalisée ou propriétaire, permettant la portabilité des données ou des applications.

5.7. Quelques avantages et inconvénients d'un Cloud [1.30]

5.7.1 Avantages

S**ouplesse d'évolution :** le client dispose souvent en temps réel des évolutions de la plateforme de Cloud computing, étant donné qu'il n'y a pas de logiciel à installer et que l'accès se fait avec un simple navigateur web.
Simplicité : l'entreprise cliente n'a plus besoin de développements coûteux et déplace la responsabilité du fonctionnement du service sur le fournisseur.

Liberté de changer : le Cloud computing étant généralement facturé à la demande ou par abonnement mensuel, il est très facile pour une entreprise d'arrêter le service si elle n'en a plus besoin ou si elle souhaite aller chez un concurrent.

Coût : la force du Cloud computing réside dans la possibilité de proposer le même service à un grand nombre d'utilisateurs, et donc de pouvoir amortir les coûts de fonctionnement sur toute la base d'utilisateurs. Au final, le coût d'une solution de Cloud computing sera donc très raisonnable par rapport à une solution sur mesure.

L'accessibilité : Les données sont sur un serveur, consultables à n'importe quel moment et où que l'on soit via une connexion Internet.

Partage et travail collaboratif : On peut également partager nos ressources et permettre ainsi un travail à plusieurs.

Fiabilité : Les services basés sur des infrastructures performantes possédant des politiques efficaces de tolérance aux pannes.

5.7.2. Inconvénients

Problème de sécurité

- Communications sensibles vers l'extérieur
- Données stockées dans le nuage

Le contrôle des données

- Localisation exacte

Fortement dépendant du réseau

- Plus de réseau, plus d'application
- Panne du fournisseur, applications inaccessible

5.8. Vers la fédération de nuages ou « Intercloud » [1.31]

Comme expliqué précédemment un nuage correspond à une infrastructure et à son domaine d'administration. De façon plus simple, il est courant d'associer un nuage à l'entreprise qui est responsable de la gestion de l'infrastructure associée. On parlera alors du nuage d'Amazon, de celui de Google, du nuage de Microsoft...etc. Cependant, du point de vue d'un utilisateur, cet

ensemble de nuages accessibles publiquement via Internet peut être vu comme un méta-nuage, au sein duquel un certain nombre de services et de ressources informatiques sont disponibles. De la même façon qu'Internet est le réseau des réseaux. Ce méta-nuage est le nuage des nuages et on l'appelle « Intercloud », son principe est illustré sur la **figure 1.12**.

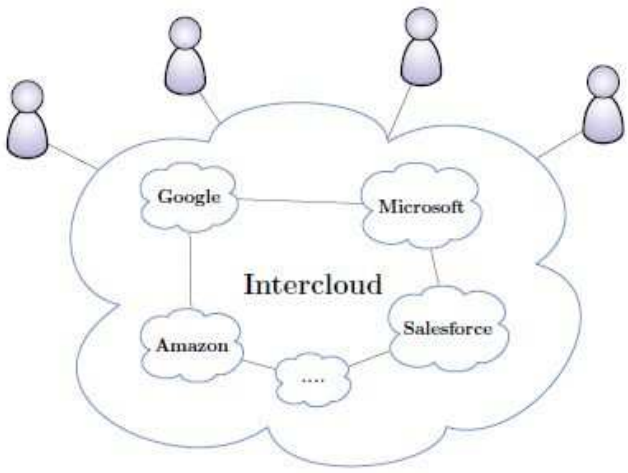


Figure 1.12. : Intercloud : le nuage des nuages [1.31].

5.9. Exemple de nuage de calcul

5.9.1. Amazon EC2 [1.32]

(Elastic Compute Cloud) est un service d'Amazon qui permet justement aux développeurs de disposer et de configurer ces ressources informatiques suivant leurs besoins. Le principe de base est relativement simple: le développeur peut créer des briques appelées des "instances EC2", une instance EC2 étant une configuration particulière de ressources : une certaine taille mémoire (RAM), une puissance de calcul, un choix d'OS (Windows server, Linux, Oracle etc.), une taille d'espace de stockage.

Ainsi si votre application doit traiter énormément de données vous pouvez privilégier une grosse puissance de calcul, s'il y a beaucoup d'échanges de données vous privilégierez la mémoire etc.… Chaque instance EC2 peut être configurée finement par le développeur qui peut

aussi les multiplier en fonction de l'importance du trafic. Le développeur a la main sur ces ressources et peut les allouer et les libérer en temps réel.

5.9.2. Amazon S3 [1.33]

C'est le service de stockage d'Amazon. Au lieu d'acheter des disques durs, grâce à Amazon S3 l'utilisateur peut consommer exactement l'espace de stockage dont il a besoin. Si son besoin augmente il achète davantage d'espace et si au contraire il diminue il peut libérer de l'espace et payer moins. L'utilisateur n'a pas à se soucier de planifier ses besoins, l'espace est disponible à la demande et en temps réel.

5.9.3. Windows Azure [1.34]

Est la solution de PaaS (Platform as a service) de Microsoft. Cette plate-forme fournit aux développeurs l'ensemble des services nécessaires pour permettre le développement et l'exécution d'applications dans le nuage. Windows Azure s'appuie sur un ensemble de Datacenter répartis autour de la planète.

Les milliers de serveurs installés dans ces centres fournissent la puissance informatique ainsi que les capacités de réseau et de stockage nécessaires à l'exécution des applications des clients, le tout sous la forme de services granulaires dont la capacité peut être adaptée à la volée en fonction de l'évolution des besoins des applications.

6. Le Sky Computing (calcul dans le ciel) [1.35]

C'est la nouvelle génération de Cloud, elle consiste a faire reliée et fédérer plusieurs nuages de calculs en les faisant travailler sous une seul et même organisation virtuelle afin de résoudre des problèmes de grande envergure et offrir des services nécessitant une haute qualité de service et rapide mise a l'échelle.

7. Conclusion

Dans ce chapitre nous avons donné un aperçu sur les notions fondamentales des systèmes distribués à grande échelle notamment le concept de grille telle que leurs objectifs, architectures, logiciels et domaines d'application puis on a donné un aperçu sur le calcul dans les nuages (Cloud Computing), ses services. On a terminé ce chapitre avec une ébauche sur le calcul dans le ciel (le Sky Computing) qui est le future du l'extension du Cloud et de d'internet.

1. Introduction

L'ordonnancement joue un rôle essentiel dans de nombreux secteurs d'activités : la conception (de bâtiments, de produits, de systèmes, . . .), l'administration (gestion d'emplois du temps, gestion du personnel), l'industrie (gestion de production), l'informatique (ordonnancement de processus). Il s'agit généralement d'organiser dans le temps l'exécution de tâches soumises à des contraintes de temps et de ressources, tout en satisfaisant au mieux un ou plusieurs objectifs. Les méthodes d'ordonnancement foisonnent dans la littérature. Elles se différencient par la nature du problème considéré (nombre de ressources, structure particulière du problème, . . .), la nature des contraintes prises en compte, les objectifs à satisfaire (minimisation des coûts, de la durée totale de mise en œuvre, . . .) et la nature de l'approche de résolution adoptée (heuristiques, méthodes exactes, approches par contraintes, . . .) **[2.1].**

Dans ce chapitre, on abordera l'ordonnancement de tâches sur une grille de calcul, on présentera quelques notions de base sur l'ordonnancement et les paragraphes qui suivent s'intéressent à l'ordonnancement des tâches dépendantes et des tâches indépendantes et leurs algorithmes.

2. Définition

Le problème d'ordonnancement consiste à organiser dans le temps la réalisation de tâches, compte tenu : des contraintes temporelles (délais,...) et des contraintes portant sur l'utilisation et la disponibilité de ressources requises **[2.2].**

Une solution d'ordonnancement décrit les dates prévues pour l'exécution des tâches et l'allocation des ressources au cours de temps. La méthode utilisée pour l'élaboration d'un ordonnanceur vise à satisfaire un où plusieurs objectifs (coût, délais, qualité) **[2.3].**

3. Vue sur le problème d'ordonnancement [2.4]

Dans un problème d'ordonnancement quatre notions fondamentales interviennent. Ce sont les tâches, les ressources, les contraintes et les objectifs.

Tâche : une tâche est définit par un ensemble d'opération qui doivent être exécutées.

Ressource : une ressource est un moyen matériel (machine) ou humain intervenant dans la réalisation d'une tâche.

Il existe deux principaux types de ressources :

- ✓ *Les ressources renouvelables* : elles sont disponibles en quantité constante tout au long de l'exécution des tâches. Les ressources renouvelables comme les machines, les processeurs, etc.
- ✓ *Les ressources consommables* **:** une ressources est consommable si, après avoir été allouée à une tâche, elle n'est plus disponible pour les tâches restant à exécuter, c'est le cas des matières premières et de l'énergie par exemple.

Contrainte : les contraintes représentent les limitations imposées par l'environnement ou les ressources.

Objectif : ce sont les critères à optimiser.

La résolution d'un problème d'ordonnancement consiste à déterminer :

- ➢ Le placement des travaux(ou jobs) dans l'espace, c'est-à-dire sur les processeurs ;
- ➢ Le placement des travaux dans le temps, c'est-à-dire les instances de début d'exécution de chaque travail sur des ressources qui participent à sa réalisation ;

Dans de nombreux problème d'ordonnancement deux hypothèses de base sont généralement respectées :

- ➢ A chaque instant, une machine ne peut exécuter qu'un seul travail ;
- ➢ A chaque instant, un travail peut être exécuté par une machine au plus.

Cependant, il y a des problèmes d'ordonnancement plus spécifique :

- ➢ L'ordonnancement par lot(ou batch), les tâches ne sont pas ordonnancées aussitôt qu'elles arrivent. Elles sont d'abord collectées dans une file d'attente, qui sera ordonnancée ultérieurement, à un instant prédéterminé.
- ➢ L'ordonnancement par chevauchement, où les opérations d'un même travail peuvent être en cours d'exécution sur plusieurs machines à un instant.

Aussi, deux modes d'exécution sont possibles :

- Avec préemption : l'exécution d'une même opération peut être interrompue puis reprise sur une des machines (migration de codes).
- Sans préemption : si une opération à commencé, elle doit être menée jusqu'a son terme sur la même machine, sans interruption.

3.1. Le problème d'ordonnancement sur une grille de calcul

Le problème de l'ordonnancement sur une grille de calcul peut être représente par la **figure 2.1**. Il s'agit d'attribuer les tâches de l'application à exécuter aux différentes ressources de la grille. Pour cela, l'ordonnanceur considère d'une part les caractéristiques de l'application, et d'autre part celles des ressources disponibles. L'ordonnancement est alors calculé de manière à satisfaire un objectif fixé **[2.5].**

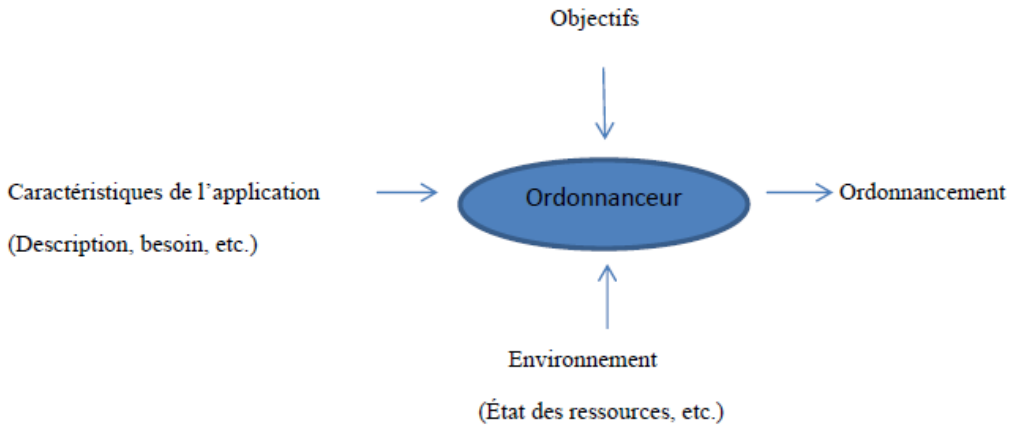


Figure 2.1 : **le problème d'ordonnancement sur une grille [2.5].**

3.1.1. Objectifs de l'ordonnancement [2.5]

L'ordonnanceur (Figure 2.1) choisit les ressources auxquelles les tâches des applications soumises seront associées de manière à atteindre un objectif. Deux types d'objectifs sont principalement considérés.

- **Objectifs centrés sur les applications :**

Ces objectifs tendent à satisfaire au mieux les exigences de chacune des applications soumises. La fonction objective à minimiser pour trouver un ordonnancement peut inclure les paramètres suivants.

- ✓ *Temps d'exécution :* le temps mis pour exécuter l'application.
- ✓ *Temps d'attente* : le temps passé par l'application dans une file d'attente, avant son exécution d'une part, et durant son exécution si elle subit des interruptions.
- ✓ *Temps de réponse* : le temps total, correspondant à la somme du temps d'exécution et du temps d'attente.

- **Objectifs centrés sur les ressources :**

Les objectifs appartenant à cette catégorie favorisent un jeu de métriques liées au système, et non aux applications. Les propriétés suivantes du système peuvent alors être considérées:

- ✓ *Capacité de traitement :* le nombre d'applications qui se terminent par unité de temps (par heure, par jour, etc.).
- ✓ *Taux d'utilisation* : le pourcentage de temps durant lequel la ressource est occupée.

3.1.2. Caractéristiques des applications [2.5]

Différentes caractéristiques des applications candidates à l'exécution sur un support tel qu'une grille de calcul doivent être prises en compte par le gestionnaire de ressources de la grille.

La première de ces caractéristiques correspond aux exigences des applications liées aux calculs. Il s'agit de fournir à l'ordonnanceur des données telles que le temps processeur nécessaire pour les exécuter, ou encore l'espace mémoire demandé. D'autres facteurs, tels que la date de début requise par l'application ou son cout maximal, peuvent être pris-en considération.

De plus, il est également possible de distinguer le cas des applications batch de celui des applications interactives. Les premières ne nécessitent aucune intervention de la part de

l'utilisateur au cours de leur exécution. Elles sont ainsi parfaitement adaptées à une exécution sur grille, et leur exécution peut être préemptée. En revanche, les applications interactives dialoguent régulièrement avec l'utilisateur. Dans ce cas, de telles pratiques sont difficilement envisageables.

Les besoins des applications en termes de réseau doivent également être pris en considération. Par exemple, s'il s'agit d'applications parallèles, les besoins en termes de bande-passante dépendent du volume de données devant être transférées entre les tâches. Les applications paramétriques n'ont quant à elles aucune exigence particulière sur ce point.

Ces caractéristiques influenceront l'ordonnancement afin de ne pas pénaliser ces applications par un mauvais choix de ressources.

Enfin, il est possible de distinguer deux derniers cas : celui des applications temps réel, pour lesquelles les tâches sont soumises à des contraintes de deadlines (dates limitées), et les applications auxquelles un niveau de qualité de service est garanti. Pour exécuter ce type d'applications dans les délais impartis, le gestionnaire de ressources doit sélectionner les ressources pour lesquelles les performances prédites sont fiables.

3.1.3 Caractéristiques des ressources [2.5]

Les ressources gérées par les systèmes étudiés dans ce chapitre peuvent être fortement hétérogènes. Ainsi, divers paramètres les concernant doivent être pris en compte lors de la phase d'ordonnancement.

La première caractéristique que nous distinguons concerne les performances des ressources. Pour des ressources de calcul (processeurs), il s'agit par exemple du nombre d'opérations pouvant être réalisées à la seconde. Dans le cas des réseaux, c'est la bande passante qui est considérée. Dans tous les cas, les capacités des ressources constituent un paramètre important de l'ordonnancement.

Les capacités des ressources sont de plus modulées par leur charge. Le gestionnaire de ressources est alors responsable de la prédiction de performances des ressources, en se basant

sur une estimation de cette charge. Plusieurs méthodes peuvent être utilisées pour ceci : l'´etablissement d'un modèle théorique, l'estimation basée sur un historique, etc.

Une autre caractéristique importante à retenir est l'aspect dédié des ressources. La majorité des grilles fonctionne avec des ressources non-dédiées aux activités générées par la grille. En d'autres termes, les applications soumises se partagent l'accès aux ressources avec des applications lancées par les utilisateurs locaux de ces ressources. Dans de telles conditions, il est difficile de prédire leurs performances, ce qui peut avoir un impact significatif sur l'ordonnancement.

Enfin, de nombreux autres paramètres pourront être pris en compte, parmi lesquels le fonctionnement en temps partagé, ou bien encore la possibilité de préempter les applications présentes sur les ressources.

3.2. Le gestionnaire de ressources [2.5]

Un gestionnaire de ressources regroupe un ensemble de composants, dont le rôle principal est de fournir les services suivants:

- Un service d'ordonnancement,
- Un service d'information,
- Un service d'exécution et d'observation

En pratique, la **figure 2.2** montre les composants mis en jeu ainsi que leurs interactions. Le rôle du gestionnaire de ressources de la grille est de s'interfacer avec les gestionnaires locaux, d'une part pour soumettre des applications aux ressources des domaines dont ceux-ci ont la responsabilité, et d'autre part pour collecter les informations provenant de ces même ressources. Ceci garantit ainsi l'autonomie des différents domaines de la grille.
De plus, au sein d'un domaine, les utilisateurs peuvent confier des tâches au gestionnaire de ressources local, mettant ainsi en avant l'aspect potentiellement non-dédié de la grille.

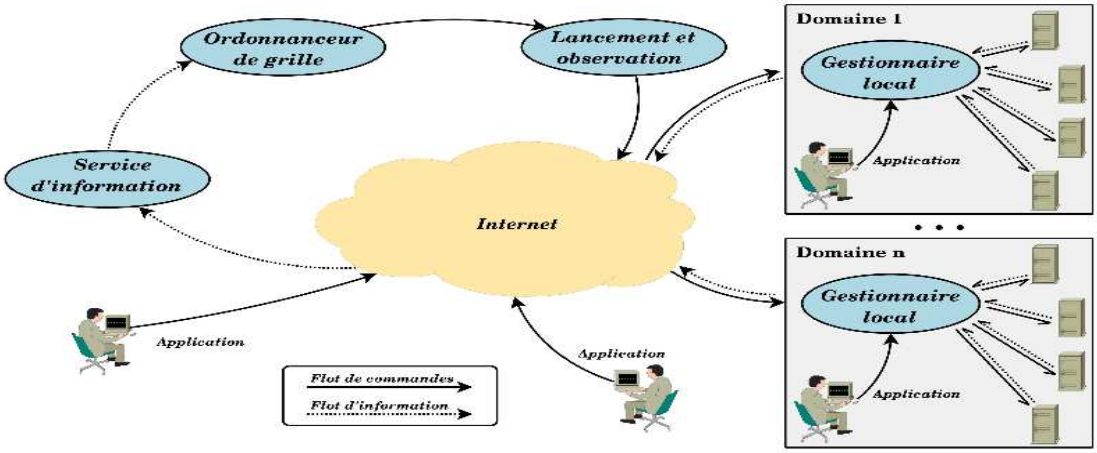


Figure 2.2 : Architecture d'un gestionnaire de grille [2.5].

L'ordonnancement sur une grille de calcul met en œuvre les trois étapes suivantes

- *Découverte des ressources :* L'utilisateur fournit au gestionnaire la description de l'application à exécuter. Le système doit alors déterminer quelles sont les ressources disponibles pour cet utilisateur. Pour cela, le service d'information est utilisé afin d'identifier les ressources présentes sur la grille. Un premier filtrage est alors opéré afin d'éliminer celles que l'utilisateur n'a pas le droit d'utiliser. De plus, le gestionnaire calcul les besoins de l'application en ressources (leur nombre, leur puissance, le temps n´nécessaire, etc.). Un second filtrage peut alors être effectué pour ne retenir que les ressources pouvant satisfaire ces besoins.
- *Sélection des ressources :* Il s'agit de l'étape d'ordonnancement. Etant donné un ensemble de ressources possibles pour exécuter l'application de l'utilisateur, le gestionnaire doit choisir celles qui seront réutilisées. Il s'appuie pour cela sur une estimation des besoins en ressources de l'application, ainsi que sur une prédiction des performances des ressources. Divers algorithmes sont alors disponibles, permettant de choisir un ordonnancement en fonction des objectifs fixés (équilibrage de charge, temps d'exécution minimal, etc.).
- *Exécution de l'application :* Cette dernière étape permet au gestionnaire de grille de se connecter à un ou plusieurs gestionnaires locaux afin de leur confier l'application à exécuter. Un suivi de la progression de cette exécution et de l'état des ressources employées peut éventuellement être mis en place.

3.3 L'allocation des ressources [2.5]

L'utilisation efficace des ressources est non seulement nécessaire du point de vue financier dans une entreprise commerciale distribuée, il est également l'une des exigences principales dans le partage de type des ressources, telles comme dans les grilles de calcul. Cependant, il est difficile de répondre aux objectifs de service d'une demande fonctionnant sur une infrastructure informatique distribuée, tout en maintenant les performances du système. La raison pour vient du fait que l'optimisation de la qualité des services offerts aux utilisateurs sur une infrastructure habituellement en conflit avec les objectifs d'efficacité des prestataires de ressources. Par ailleurs, les infrastructures distribuées sont partagées par plusieurs applications qui appartiennent à des différents domaines administratifs. Il y a beaucoup de moyens pour réserver les ressources nécessaires pour exécuter l'application sur les infrastructures distribuées à travers le middleware et les gestionnaires des ressources basée sur une politique d'ordonnancement.

3.3.1 L'ordonnancement par lot [2.5]

Le gestionnaire le plus utilisé dans les systèmes traditionnels est l'*ordonnancement par lot.* Comme il est représenté dans la **figure 2.3,** un ordonnancement par lot reçoit séquentiellement des soumissions de tâches. Il ordonnance l'exécution de ces tâches sur les ressources qu'il dirige, en déterminant quand et où chaque tâche sera exécutée. Par conséquent, les ressources sont allouées avec l'arrivée de tâches et elles sont accédées ou moment de la soumission des tâches. Comparé avec les problèmes de l'ordonnancement statiques qui présument la connaissance parfaite de l'état du système, l'exécution courante et l'échange entre tâches complètement prévisibles, les vrais systèmes par lot doivent faire face aux arrivées périodiques de nouvelles tâches dans la file d'attente et des résultats incertitudes tel que la durée exacte des tâches. Ils doivent mettre à jour leur ordonnancement dynamiquement en conséquent. La politique la plus simple d'ordonnancement par lots est First In First Out (FIFO). Avec cette stratégie naïve, il y a une possibilité qu'une tâche doit attendre depuis longtemps dans la file avant d'être exécuté. Une tâche très courte localisée dans la fin doit attendre jusqu'à ce que toutes les tâches soient localisées, avant elle. La priorité de la tâche est déterminée par les propriétés de la tâche telle que les exigences de la ressource demandées et le temps qu'il a

attendu dans la file. Ces propriétés sont combinées dans une formule avec des poids spécifiés par l'administrateur du système. Après la priorité de toutes les tâches dans la file est calculée, les ressources sont allouées à la tâche d'après son ordre de priorité. Les priorités doivent être périodiquement mises à jour pour prendre en considération des nouvelles tâches et le temps réel de l'attente.

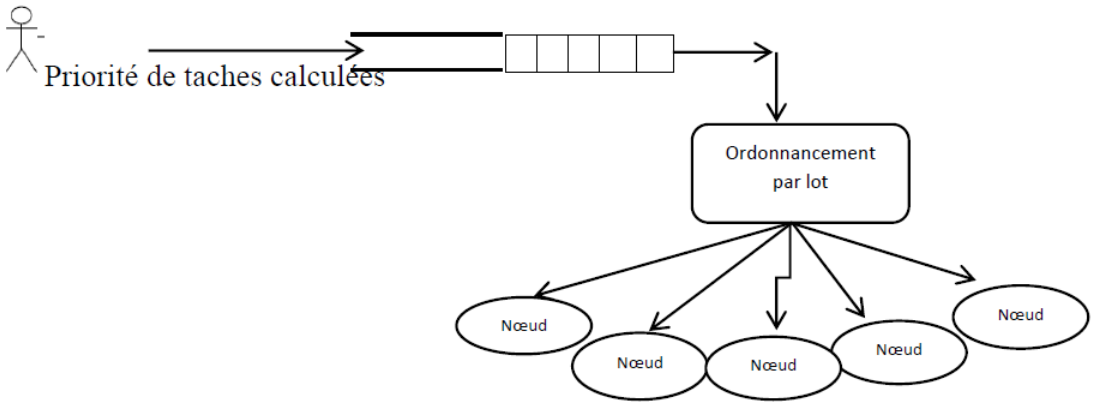


Figure 2.3 : Ordonnancement par lot [2.5].

3.3.2 Le méta-ordonnancement

Les systèmes par lot sont souvent utilisés au niveau local où les ressources sont homogènes et le nombre de ressources est limité. Un seul gestionnaire peut contrôler toutes les ressources disponibles. Les tâches sont allouées directement de la file du lot.

Un méta-ordonnanceur présenté dans la **figure 2.4** fournit un mécanisme d'ordonnancement multi-niveau pour s'adapter à l'hétérogénéité des ressources sur les infrastructures multi-sites, le méta-ordonnanceur, gère alors l'ordre de priorité des applications en fonction de la file où elles se trouvent **[2.5].**

L'ordonnancement par lot et le Méta-ordonnancement implémentent effectivement un modèle composé de trois phases :

- ✓ rassembler le statut des ressources pour l'infrastructure entière;
- ✓ décider l'allocation des tâches aux ressources;
- ✓ soumettre des tâches aux ressources.

Dans la première phase, toute l'information à propos du système doit être publiée. Dans une grande infrastructure, c'est souvent irréaliste, et l'information sera souvent non disponible, inexacte, ou hors de date. La dimension de ce problème augmente avec le nombre de tâches dans la file et la disponibilité des ressources. Les ordonnanceurs peuvent être surchargés et peuvent causés des délais dans le processus d'ordonnancement **[2.6]**

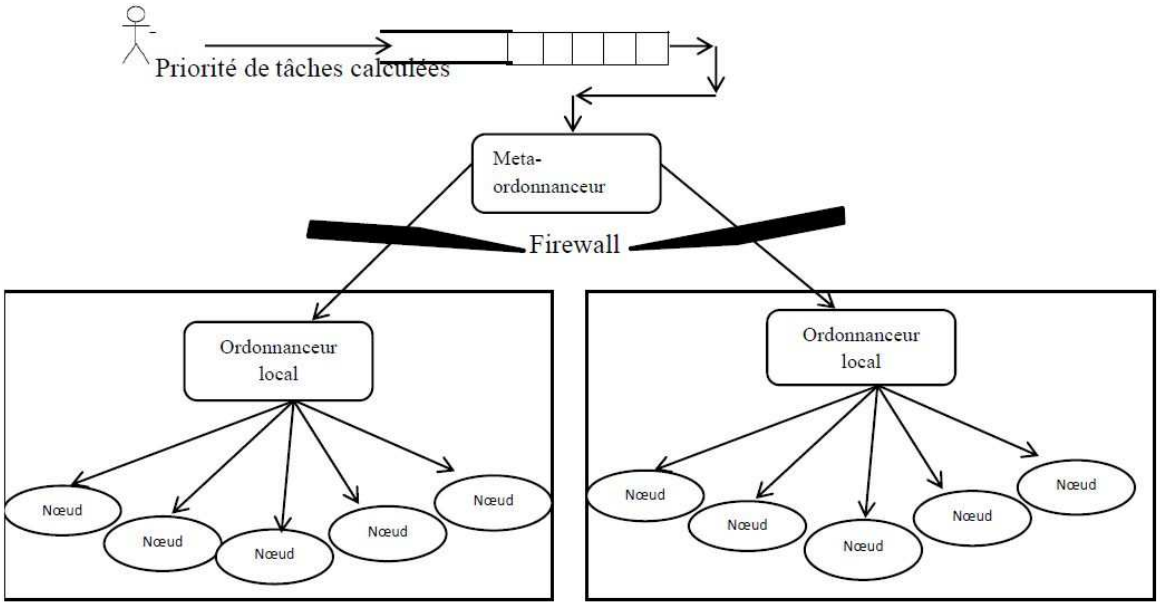


Figure 2.4 : Méta-ordonnanceur. [2.6]

3.4. Ordonnancement adaptatif [2.7]

L'objectif visé par l'ordonnancement adaptatif est de choisir la meilleure reconfiguration de l'architecture (matérielle et logicielle) et de l'algorithme de contrôle qui nous assure l'optimisation des performances de l'application par rapport aux défauts des composants.

Dans une grille de calcul, la demande pour un ordonnancement adaptatif vient de trois points: l'hétérogénéité de ressources disponibles, le dynamisme de performance de la ressource, et la diversité d'applications, comme la **figure 2.5** suivante :

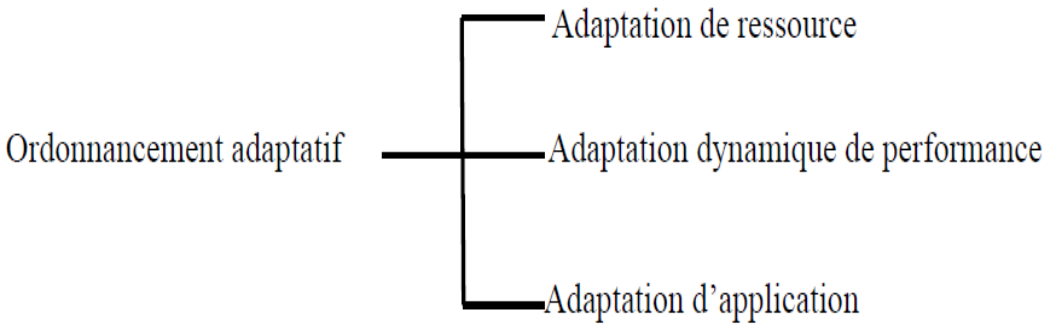


Figure 2.5 : Taxonomie des algorithmes d'ordonnancement adaptatifs dans les grilles de calcul [2.7].

3.4.1. Adaptation des ressources

En raison de l'hétérogénéité et la diversité des applications, découvrir les ressources disponibles et sélectionner un sous-ensemble approprié à l'application sont très importants pour atteindre la haute performance ou réduire le coût **[2.7]**.

3.4.2. Adaptation dynamique de performance [2.8]

L'adaptation à la performance dynamique des ressources est principalement exposée comme suit :

- ✓ Le changement des politiques d'ordonnancement ou de ré-ordonnancement (par exemple, la commutation entre les algorithmes d'ordonnancement statiques qui utilisent des informations prédites sur les ressources qui équilibrent les résultats d'ordonnancement statiques).
- ✓ La charge du travail distribué selon des modèles de performance spécifiques à l'application
- ✓ trouver un nombre adéquat de ressources qui peut être utilisé.

3.4.3. Adaptation d'application

Pour atteindre un niveau élevé de performance, les ordonnanceurs au niveau des applications dans la grille sont généralement intégrées étroitement avec l'application elle-même et ne sont

pas facilement appliquée à d'autres applications. En conséquence, chaque ordonnanceur est spécifique à une application **[2.8]**.

4. Algorithmes d'ordonnancement pour des grilles

4.1. Introduction

Les algorithmes d'ordonnancement sont considérés comme étant capable d'effectuer des calculs beaucoup plus sophistiqués. Par exemple, un algorithme d'ordonnancement peut prendre la décision d'ordonnancement actuel, basé sur l'histoire entière de travaux antérieurs et les décisions d'ordonnancement. En revanche, les politiques d'ordonnancement sont considérés comme beaucoup plus simple, et prendre des décisions d'ordonnancement basé sur des règles simples qui sont faciles à appliquer [**2.9]**.

4.2. Catégories d'ordonnancement

Des études ont été menées afin de parvenir à définir des catégories d'algorithmes d'ordonnancement **[2.5]**. Nous choisissons de retenir les catégories présentées en **figure 2.6**.

.

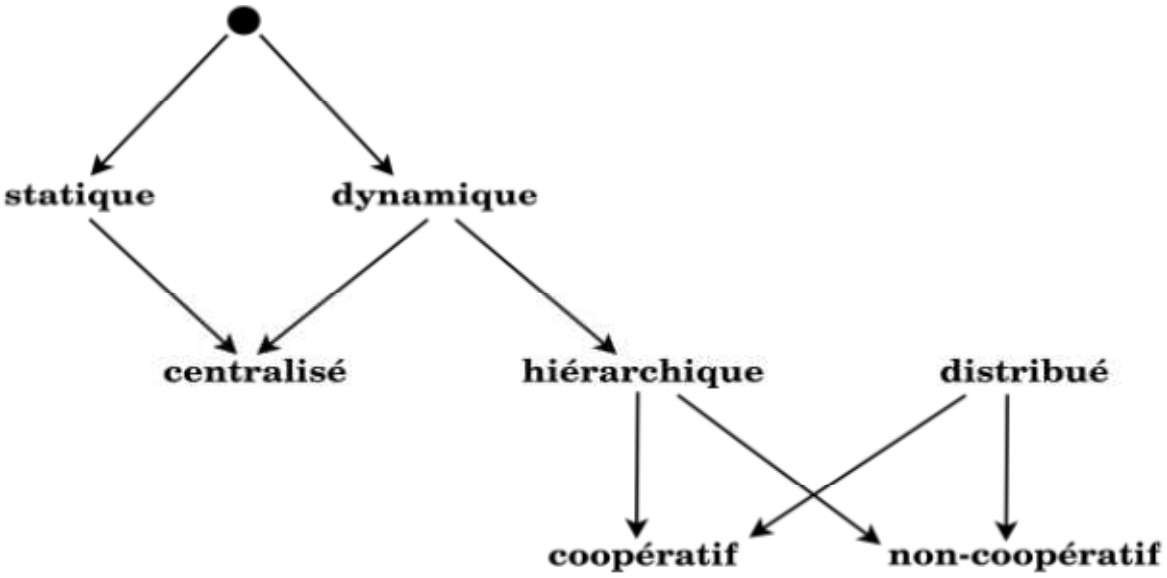


Figure 2.6: Catégories d'ordonnancements adaptés aux grilles de calcul [2.5].

4.2.1. Placement statique et placement dynamique

✓ **Placement statique**

Dans le cas du placement statique, la décision de placement est prise hors ligne, c'est à dire avant l'exécution du programme. Les informations concernant l'état des ressources ainsi que de l'application soumise sont supposées disponibles au moment où l'ordonnancement est calculé. Ce type de placement est bien adapté aux applications déterministes dont le comportement est bien maitrisé d'une part, et d'autre part aux grilles pour lesquelles l'état des ressources peut être prédit de façon fiable, comme c'est le cas des grilles dont les ressources sont dédiées **[2.5]**.

✓ **Placement dynamique**

Le placement dynamique repose sur le principe d'une allocation des tâches à la volée, lorsque l'application s'exécute. Il s'agit ainsi d'un placement en ligne. L'algorithme peut éventuellement s'adapter au cours de l'exécution de l'application à l'état des ressources, notamment à la charge des processeurs, en choisissant de migrer certaines tâches d'une ressource à une autre. Le placement dynamique est utile lorsqu'il est impossible, par exemple, de déterminer le temps d'exécution de l'application à exécuter **[2.5]**.

4.2.2. Placement centralisé, décentralisé ou hiérarchique [2.5]

Une stratégie d'ordonnancement centralisé concentre toutes les prises de décisions à un seul endroit : un ordonnanceur unique est responsable de calculer un placement des applications soumises sur l'ensemble des ressources disponibles de la grille.

Cette approche est optimale, puisque l'ordonnanceur a une connaissance de l'état de toutes les ressources. Cependant, trois points négatifs se posent :

- le passage à l'échelle n'est pas facilité par une telle approche,
- ce système n'admet pas de tolérance aux fautes,
- le goulot d'étranglement ainsi crée peut engendrer une baisse importante des performances du système.

Le modèle décentralisé, ou distribue, confie la responsabilité de l'ordonnancement a différents ordonnanceurs, agissant de manière coopérative ou non. Enfin, la dernière stratégie

d'ordonnancement repose sur un modèle hiérarchique, pour lequel les ordonnanceurs gèrent des entités de plus ou moins de hauts niveaux selon leur position dans la hiérarchie.
Ce dernier modèle constitue une combinaison de l'approche statique et de l'approche dynamique.

4.2.3. Modèle coopératif et modèle non coopératif

Dans le cas d'une stratégie d'ordonnancement dynamique ou hiérarchique, les différents ordonnanceurs peuvent collaborer entre eux pour déterminer un placement, ou bien ils peuvent travailler de manière totalement indépendante. Le modèle coopératif permet à chaque ordonnanceur de calculer une partie du placement global d'une application, en communiquant avec les autres ordonnanceurs dans le but d'évoluer vers un objectif commun.

Au contraire, les ordonnanceurs évoluant en mode non coopératif sont des entités autonomes qui prennent des décisions dans le but d'optimiser uniquement leurs propres objectifs, sans se soucier de la vue d'ensemble du système **[2.5]**.

4.3. Caractéristiques de l'ordonnancement sur grille [2.5]

Les ressources présentes sur une grille de calcul possèdent des caractéristiques devant être prises en charge par les systèmes responsables de leur gestion. Le type de grille que l'on considère (grille de calcul, d'information, de stockage, etc.) conditionne le type de ressources à gérer. Concernant une grille de calcul, il s'agit principalement des processeurs fournissant la puissance de calcul, et du réseau reliant les différents nœuds.

Les systèmes d'ordonnancement doivent prendre en considération les caractéristiques suivantes:

- *Hétérogénéité des ressources :* La grille est une interconnexion de ressources appartenant à différents domaines. Si les ressources peuvent être homogènes au sein d'un même domaine, il en est tout autrement lorsque l'ensemble de la grille est considère. De fortes différences peuvent être constatées en termes d'architectures matérielles des machines partagées, de vitesse des processeurs, de systèmes d'exploitation utilisés, de bande-passante reliant les machines au

réseau, etc. Ceci résulte en une différenciation des capacités de calcul et d'accès aux données pour chaque ressource.

- *Autonomie des sites :* Les sites constituant les grilles appartiennent à différents domaines d'administration, chacun possédant ses propres politiques d'accès, de sécurité et de gestion des ressources. Le gestionnaire de ressources doit savoir s'adapter à ces spécificités, et laisser une totale autonomie aux différents sites.
- *Diversité des applications* : Une grande diversité d'utilisateurs venant de différents horizons peuvent être amenés à utiliser la grille. Il en résulte une grande palette d'applications pouvant être exécutées, et donc devant être prises en charge par les algorithmes d'ordonnancement. Il peut s'agir d'applications parallèles communicantes qui possèdent un nombre important de tâches échangeant une quantité variable de données, de même que des applications paramétriques, pour lesquelles différentes instances indépendantes de l'application sont lancées pour des paramètres différents.
- *Ressources non-dédiées* : Les ressources présentes sur la grille ne sont généralement pas dédiées à celle-ci. Les utilisateurs des divers domaines peuvent utiliser les ressources leur appartenant sans passer par le gestionnaire de ressources de la grille, ce qui implique une diversité de nouveaux paramètres à prendre en compte. Cependant, certaines grilles de calcul possèdent des ressources dédiées, comme les grilles orientées vers la prestation de services applicatifs (ASP pour Application Service Provider).
- *Aspects dynamiques* : Une des caractéristiques des grilles de calcul est le comportement dynamique des ressources qui la composent. D'une part, la disponibilité de ces dernières peut varier fortement au cours du temps : une ressource peut rejoindre ou quitter la grille inopinément. De plus, les performances des ressources peuvent également fluctuer en fonction de l'importance de leur utilisation

Les caractéristiques ainsi mises en avant sont très différentes de celles que l'on retrouve dans les systèmes traditionnels. Par exemple, les ressources d'un cluster sont généralement homogènes, et peuvent être considéré comme stables. Il est par conséquent nécessaire d'adapter les gestionnaires de ressources afin de prendre en compte l'ensemble de ces caractéristiques.

4.4. La dépendance des tâches d'une application

Les dépendances de tâches permettent de définir le moment où une tâche peut débuter. Par exemple, l'éclairage ne peut être installé que lorsque l'installation du filage est complétée, De plus, un temps d'attente, c'est-à-dire un intervalle, peut être défini entre les tâches dépendantes, afin de respecter les délais requis **[2.3].**

La dépendance à un impact crucial pour la conception des algorithmes d'ordonnancement, donc dans cette subdivision, les algorithmes sont discutés en suivant la même dichotomie comme fait illustrer dans la **figure 2.7** suivante: **[2.2]**

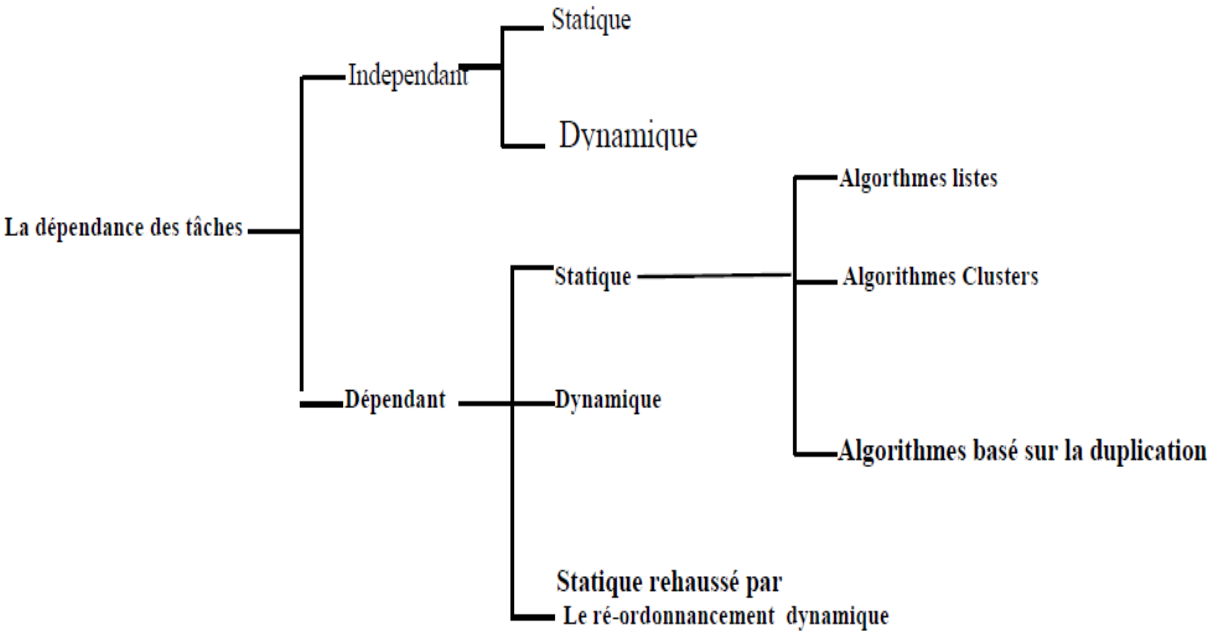


Figure 2.7 : Les algorithmes d'ordonnancement de tâches [2.2].

4.4.1. Ordonnancement des tâches indépendantes

Dans ce type d'ordonnancement, toutes les informations sont connues à l'avance comme leur temps d'exécution sur les différentes ressources, etc. Dans ce cas, les tâches s'exécutent en parallèle, indépendamment les unes des autres. Du point de vue des applications, les algorithmes heuristiques statiques fondés sur l'exécution et l'estimation des coûts peuvent être appliqué [**2.3].**

Dans ce qui suit, on présente les principaux algorithmes consacrés aux tâches indépendantes

4.4.1.1. Algorithmes d'ordonnancement des tâches indépendantes

Avant d'entamer cette partie on donne des définitions de deux termes utilisés dans les paragraphes qui suivent :

Temps de complétion maximal (makespan) : Le temps de complétion d'une application est le temps compris entre la date d'exécution de la première tâche d'un travail à exécuter et la date de terminaison de la dernière tâche.

Optimiser un ordonnancement suivant ce critère concourt à maximiser l'utilisation des ressources car cela revient à exécuter les travaux dans un temps le plus court. Ce critère est adapté à l'ordonnancement de tâches indépendantes ou de graphes de tâches connues à l'avance **[2.3].**

***Heuristique* :** En informatique, une heuristique est une technique visant à accélérer la recherche d'une solution à un problème :

- ✓ Son but est d'aider à chercher dans la bonne direction
- ✓ Fait souvent appel à des connaissances "expertes"

Les techniques heuristiques permettent généralement de faire un compromis entre la rapidité de la recherche et la qualité de la solution trouvée :

- ✓ Trouver une solution optimale en un temps "pas trop long"
- ✓ Trouver rapidement une solution "pas trop mauvaise" **[2.9].**

Dans la littérature existe plusieurs algorithmes d'ordonnancement de tâches indépendantes dans les paragraphes qui suivent, on introduit les plus connus.

- ✓ **Équilibrage De Charge Opportuniste (Opportunistic Load Balancing (OLB))**

Dans cette ancienne méthode, la machine qui est inactif est sélectionnée sans tenir compte du temps d'exécution de la tâche. Si deux ou plusieurs machines sont au repos alors la machine est choisie arbitrairement. Dans cette méthode la période requise pour l'ordonnancement est moins et il garde presque toutes les machines occupées tout le temps possible **[2.10]**.

✓ **Temps D'Exécution Minimum (MET : Minimum Execution Time)**

MET attribue chaque tâche à une ressource avec le meilleur temps d'exécution prévu pour cette tâche, peu importe si cette ressource est disponible ou non au temps demandé. La motivation derrière MET est de donner à chaque tâche sa meilleure machine. Ceci peut provoquer un déséquilibre de charge sévère dans les machines **[2.10]**.

✓ **Temps Minimum D'Accomplissement (MCT : Minimum Completion Time)**

MCT attribue à chaque tâche, dans un ordre arbitraire, une ressource avec un temps d'achèvement minimum pour cette tâche. Cela provoque une certaine affectation tâche à des machines qui n'ont pas le temps d'exécution minimum pour elles. L'intuition derrière MCT est de combiner les avantages de l'équilibrage de chargement opportuniste (OLB) et MET **[2.10]**.

✓ **Algorithme De Commutation (SA : Switching Algorithm)**

Cette méthode d'ordonnancement combine les meilleures caractéristiques de MCT et MET de méthodes de l'ordonnancement. La méthode essaie d'utiliser mieux la charge d'équilibrage des MCT et l'exécution sur la machine la plus rapide du MET. Ici, l'idée est d'utiliser d'abord le MCT jusqu'à un seuil d'équilibre est atteint suivie par MET, ce qui crée le déséquilibre de charge par l'attribution de tâches sur des machines plus rapides **[2.10]**.

✓ **k-Percent Best (kPB)**

Cette méthode tente également de combiner les meilleures caractéristiques de MCT et MET simultanément au lieu de façon cyclique. En assignant les tâches à k meilleures ressources parmi 100 en fonction de temps d'exécution des tâches. Pour chaque tâche la ressource qui donne le minimum temps de complétion est sélectionnée **[2.11].**

Cette méthode a un inconvénient si ces k ressource sont occupée alors y ' a d'autre ressource libre.

✓ **Min_min**

C'est une heuristique qui combine à la fois MET et MCT. Il sélectionne la tâche ayant le moins de temps exécution ensuite il assigne la tâche en utilisant MCT pour attribué la tâche à la ressource prévu pour la compléter au plus tôt **[2.12]**.

Algorithme1 : Min-min [2.13]

Pour toutes les tâches Ti dans la liste des tâches L

Pour toute machine Mj

Eij = date_courante + Cij (Cij : temps d'exécution du tâche Ti)

Fin pour

Fin pour

Tant qu'il existe des tâches non encore ordonnancer

Pour toutes les tâches Ti avec i = 1..n dans L

Pour toutes les machines Mj avec j = 1..m

Trouver le minimum des dates de fin d'exécution Eij

Fin pour

Fin pour

Soit (Ta, Ma) respectivement la tâches et la machine qui correspondent au minimum

Affecter la tâches Ta à la machines Ma

Retirer la tâches Ta de la liste L

Mettre à jour les dates de disponibilités des processeurs de machines Ma

Trier les processeurs de la machines Ma dans l'ordre croissant de leur dates de disponibilités

Mettre à jour la date Ei, Ma pour toute Ti dans L

Fin Tant que

✓ **Max-min**

L'heuristique Max-min est similaire à l'heuristique Min-Min. Au lieu de choisir la plus petite tâche il sélectionne la tâche la plus grande en premier ensuite il applique MCT pour attribué la tâche à la ressource prévu pour la compléter au plus tôt **[2.12]**.

Algorithme2 : Max-min [2.13]

Pour toutes les tâches Ti dans la liste des tâches L

Pour toutes machines Mj

Eij = date_courantes + Cij (date_courantes est la date du système)

Fin pour

Fin pour

Tan que il existe des tâches non encore ordonnancée

Trouver le minimum des dates fin d'exécution pour chaque tâche

Trouver le maximum de ces dates de fin d'exécution

Soit (Ta, Ma) respectivement la tâche et la machine choisies

Affecter la tâche Ta à la machine Ma

Retirer la tâche Ta de la liste L

Mettre à jour les dates de disponibilité des processeurs dans la machine Ma

Trier les processeurs de la machine Ma dans l'ordre croissant de leurs dates de Disponibilités

Mettre à jour la dates *Ei, Ma* pour toute tâche Ti dans L

Fin Tan que

- ✓ **LJFR-SJFR**

Consiste à affecter les grandes tâches sur les ressources les plus rapides, Courtes tâches sur la plus rapide de ressources. (LJFR-SJFR) alloue la plus grand tâche pour la ressource la plus rapide pour réduire le makespan et alloue la petite tâche pour la plus rapide ressource afin de réduire le temps d'écoulement d'ordonnancement **[2.11]**.

Dans la première étape les m tâches sont allouées à les m ressource disponible en utilisant les étapes de la méthode max-min. Dans la deuxième étape les n-m tâches sont affectés par la

méthode min-min et max-min d'une manière alternative (la plus petite tâche sur la ressource la plus rapide puis la plus grande tâche sur la plus rapide ressource **[2.11].**

✓ **Sufferage**

Une autre heuristique bien connue qui est basée également sur MCT est la Sufferage. Le raisonnement derrière cette heuristique est qu'une tâche doit être affectée au processeur qui la rendra moins souffrir. La Sufferage est défini en tant que la différence entre la meilleure MCT et la deuxième meilleure MCT. Lors d'ordonnancement, l'algorithme donne la plus grande priorité aux tâches qui ont la plus grande valeur Sufferage **[2.13]**.

Algorithme Sufferage [2.13]

Pour toutes les tâches Ti dans la liste des tâches L

Pour toutes machines Mj

Eij = date_courantes + Cij (date_courantes est la date du système)

Fin pour

Fin pour

Tan que il existe des tâches non encore ordonnancée

Pour toutes tâche Ti dans L

Trouver le premier minimum (min1i) et le deuxième minimum (min2i) des Eij

Suffragei = min2i – min1i

Trouver respectivement la tâche Ta et la machine Ma qui donne le maximum de suffragei

Affecter la tâche Ta à la machine correspondante

Retirer la tâche Ta de la liste L

Mettre à jour les dates de disponibilité des processeurs dans la machine Ma

Trier les processeurs de la machine Ma dans l'ordre croissant de leurs dates de disponibilités Mettre à jour la dates Ei, Ma pour toute tâche Ti dans L

Fin tant que

✓ **Work Queue (WQ)**

. Les tâches sont choisies aléatoirement dans la liste des tâches non assignées et affectée à la machine avec un minimum charge de travail. Affectation de tâches répétées de la même manière jusqu'à ce que la liste des tâches non affectés s'épuise **[2.10]**.

4.4.2. Ordonnancement des tâches dépendantes

4.4.2.1. Définition d'un DAG (graphe acyclique dirigé)

Un DAG est un ensemble de N nœuds qui sont connectés par des arcs *dirigés* : il n'y a aucun chemin dirigé dans le graphique des tâches qui peut commencer et finir à un nœud particulier n et *acyclique* veut dire qu'il n'y pas de cycles dans n'importe quel chemin du graphe. Chaque nœud a un poids nommé coût de calcul et noté par **C (n) [2.14].**

Les études sur l'ordonnancement d'applications composées de tâches dépendantes visent à fournir une stratégie pour l'ordre et la carte de workflow des tâches sur les ressources hétérogènes basés sur l'analyse des dépendances du graphe de tâches entier, afin de compléter ces tâches interdépendantes au plus tôt. De nombreuses solutions d'ordonnancement proposées sont des heuristiques basées sur les poids des nœuds de la tâche et des arêtes d'un graphe de tâches **[2.14].**

Comme illustré dans la **figure 2.7**, un poids wi est affectée à une tâche Ti et un poids (wi, j) est associé à un bord (Ti, Tj).

Wi : Temps d'exécution de la tâche Ti.
Wi,j : Temps de communication entre les tâches Ti et Tj.

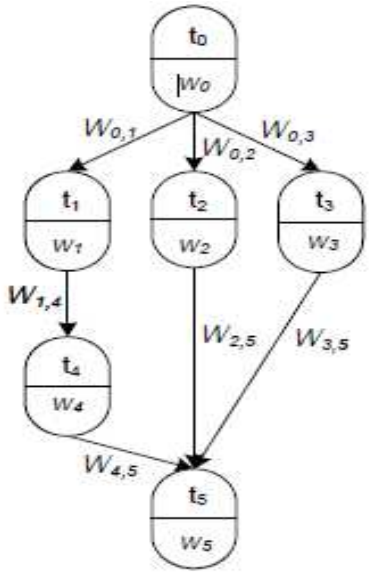


Figure 2.8 : Un exemple de graphe de tâches dépendantes [2.14].

4.4.2.2. Définition d'un Workflow

Afin d'automatiser les traitements et gagner du temps, les scientifiques ont besoin de rassembler leurs tâches logicielles et de les connecter ensemble pour former une application. À l'entrée de l'application, des données commencent à être traitées par les tâches qui n'ont pas de dépendance, puis celles-ci transmettent les résultats aux suivantes. Généralement la structure d'une telle application se présente sous la forme d'un DAG (Directed Acyclic Graph : graphe orienté sans cycle) : un graphe dont chaque sommet est une tâche et chaque arc est une contrainte de dépendance. Une dépendance modélise également une communication entre deux tâches **[2.15].**

Ceci nous amène à la notion de workflow. Dans la littérature la notion de workflow est généralement définie comme une structure abstraite d'une application décrite par un DAG. Les travaux de recherche distinguent l'exécution d'une application munie d'une structure workflow qui réalise un traitement sur des données d'entrée et l'exécution en parallèle de plusieurs instances d'une application munie d'une structure workflow qui réalise un traitement sur un ensemble de données d'entrée. Une autre définition du terme workflow est un ensemble d'instances d'une ou plusieurs applications décrites par un DAG **[2.16].**

4.4.3. Les algorithmes d'ordonnancement des tâches dépendantes

Dans ce qui suit, nous détaillons les algorithmes statiques, ces algorithmes d'ordonnancement des tâches dépendantes peuvent être classés en diverses catégories. Selon la méthode qu'ils utilisent, on distingue 3 catégories: ***L'algorithme heuristique liste, duplication des tâches et L'heuristique clustérisé***.

4.4.3.1. L'heuristique liste [2.17]

Les heuristiques de liste maintiennent une liste de tâches triées suivant une priorité. Ces heuristiques sont toutes basées sur la succession des deux étapes suivantes :

1. Prendre une tâche parmi celles qui sont prêtes et/ou pas encore allouées à une machine. Une tâche est prête lorsque toutes les tâches parentes sont terminées et les données nécessaires aux calculs sont disponibles.
2. Sélectionner une machine pour exécuter la tâche et allouer celle-ci à la machine.

La première étape est aussi nommée phase de *prioritisation* car elle permet de faire un choix lorsqu'il existe plusieurs tâches disponibles. La priorité attribuée aux tâches permet de donner un ordre d'exécution. Cet ordre est déjà induit par l'ordre partiel défini par la relation de précédence qui se retrouve avec une traversée topologique du graphe de tâches. Deux types d'algorithmes de listes se distinguent. Les heuristiques à priorité statique : la priorité pour la tâche est établie une fois pour toute, et ne sera jamais remise en cause au cours de l'heuristique. A l'inverse, les heuristiques dynamiques réévaluent la priorité d'une tâche au fur et à mesure de l'exécution du DAG.

Exemples de cette heuristique

- ✓ **HEFT (*Heterogeneous-Earliest-Finish-Time*)**

Parmi les algorithmes d'ordonnancements de liste les plus répandus, celui ayant un des meilleurs rapports performances-complexité est HEFT (Heterogeneous Earliest Finish Time). Cette heuristique proposée par Topcuoglu et al **[2.18]** détermine un ordonnancement totalement statique d'un DAG sur un environnement hétérogène de manière à minimiser la durée totale d'exécution de l'application (makespan) **[2.17].**

Dans le détail, l'algorithme est constitué de deux phases principales :

- ✓ Détermination des priorités ;
- ✓ Sélection du processeur (machine).

La première étape consiste donc à affecter les dites priorités aux tâches à ordonnancer. HEFT définit la priorité d'une tâche comme la longueur du chemin le plus long partant de celle-ci (en prenant en compte les temps d'exécution et les temps de communication). La seconde étape peut alors s'effectuer comme suit : lorsqu'une tâche est retirée de la liste, l'algorithme recherche le processeur capable de minimiser sa date de fin – y compris en tirant parti des temps d'inactivité laissés à ce stade par l'ordonnanceur (insertion) – et la lui alloue dans la mesure où toutes les dépendances de données sont satisfaites **[2.19].**

✓ Earliest Task First (ETF) [2.20]

L'objectif de cette heuristique est de minimiser les temps d'inactivités des processeurs à un instant donné, sans même se soucier de l'impact de ces choix sur la suite de l'exécution.

Ainsi lorsqu'un processeur devient inactif, l'algorithme sélectionne la prochaine tâche à exécuter. Pour chacune des tâches prêtes, l'algorithme calcule la date à laquelle la tâche considérée peut être exécutée le plus tôt possible. L'algorithme considère la date minimale avant laquelle aucune tâche ne peut être exécutée, Si un processeur devient inactif avant cette date, l'horloge virtuelle est avancée au prochain passage d'un processeur à l'état inactif. La liste de tâches prêtes est mise à jour et l'algorithme recommence le processus de sélection.

❖ L'hétérogénéité dans l'heuristique liste

Une question cruciale dans les heuristiques de liste pour les DAG est de savoir comment calculer le rang d'un nœud. Dans un environnement hétérogène, le temps d'exécution de la même tâche sera différent sur différents ressources ainsi que le coût de communication via des liaisons de réseau différentes. Donc, pour un nœud particulier, son rang sera également différent s'il est affecté à des ressources différentes. Le problème est comment choisir la bonne valeur utilisée pour prendre la décision de commander. Ces valeurs peuvent être la valeur moyenne, la valeur médiane, la pire valeur, la valeur optimale et ainsi de suite. Mais Zhao et al

[2.20] ont montré que des choix différents peuvent affecter la performance des heuristiques de liste comme HEFT de façon spectaculaire (makespan peut modifier 47,2% pour certain graphe). Motivé par cette observation, Sakellariou et al **[2.21]** ont donné un hybride algorithme qui est moins sensible à des approches différentes pour les rangs des nœuds. Dans cet algorithme, les tâches sont classées et triés d'une façon décroissante. Ensuite, les tâches regroupées dans des groupes et dans chaque groupe, les tâches sont indépendantes. Enfin, chaque groupe peut être affecté à des ressources en utilisant des heuristiques pour des tâches indépendantes. Les algorithmes ci-dessus ont exploré comment l'hétérogénéité des ressources et des tâches à un impact sur l'algorithme d'ordonnancement, mais ils ne considèrent l'hétérogénéité des ressources et ratent l'hétérogénéité des liens de communication.

4.3.3.2. Algorithmes basés sur la Duplication

Les heuristiques à duplication de tâches consistent à allouer de manière redondante certaine tâches « importantes » dont d'autres dépendent. Le but recherché est donc de réduire le temps avant que les tâches en attente puissent commencer ce qui peut éventuellement améliorer le temps d'exécution global de l'application **[2.4].**

Exemples de cette heuristique :

- ✓ **TANH (Task duplication Algorithm for Network of Heterogeneous system)**

Cet algorithme consiste en quatre phases. Dans la première étape de cet algorithme, le graphe est traversé de manière Top-Down pour calculer les dates au plus tôt de début et de _n d'exécution. Puis au niveau de chaque nœud, le processeur favori et le prédécesseur favori sont déterminés pour les tâches. Dans une deuxième étape, les temps de début et de fin au plus tard sont déterminés de façon bottom-up. Le temps de fin au plus tard d'un nœud est utilisé pour calculer le temps de début au plus tard d'un successeur de ce nœud. Les autres étapes de l'algorithme sont la génération des grappes et le choix des processeurs pour leur exécution. **[2.4].**

✓ **TDS (Task Duplication-based Scheduling**

Qui ordonnance un DAG sur des processeurs hétérogènes interconnectés par des liens homogènes de communication. L'heuristique est composée de trois étapes. Initialement, pour chaque tâche du graphe les temps de début et de fin au plus tôt sont calculés en ordre de manière à identifier le prédécesseur qui devrait être ordonnancé avec la tâche et le processeur qui réduisent au maximum son temps de fin. La prochaine étape consiste à ordonnancer les tâches (dans l'ordre de leur niveau), basée sur le prédécesseur et le processeur favori, et sur les temps de début et de fin au plus tard de la tâche sélectionnée. Dans la dernière étape, une procédure de duplication tente de reproduire le prédécesseur favori de la tâche sélectionnée sur son processeur si le makespan diminue **[2.4].**

4.3.3.3. Heuristique de clustering

Les algorithmes de regroupement ou clustering sont composé typiquement de deux étapes. Durant la première étape, les tâches sont regroupées ensemble avec leurs prédécesseurs, afin de réduire les coûts de communication. La deuxième étape alloue les groupes résultants aux processeurs disponibles de système **[2.4].**

Exemple de cette heuristique

✓ **CTHP (Clustering Tasks onto Heterogeneous Processors)**

Cet algorithme est constitué de deux étapes. Initialement la stratégie suppose un environnement virtuel homogène tout en créant des groupes de tâches en utilisant la structure de l'algorithme de regroupement. Dans une deuxième étape, un sous-ensemble des clusters déjà générés sont plaqués à l'environnement hétérogène, compte tenu du coût d'exécution des tâches sur les processeurs et des caractéristiques de transmission sur les liens de communication **[2.4].**

❖ **L'hétérogénéité dans l'heuristique clustérisée [2.14]**

De toute évidence, la recherche ne considère pas ce problème, Heuristiques Cluster ne sont pas encore amélioré pour les grilles de calcul, ou la communication est coûteuse et la

performance des ressources varie dans temps. Par conséquent, cette heuristique reste un sujet intéressant pour la recherche dans l'environnement les grilles de calcul. Une autre valeur de l'heuristique de cluster pour l'ordonnancement de grille est sa nature multi-phase, qui offre plus de flexibilité aux ordonnanceurs de grille d'employer des stratégies différentes en fonction de la configuration et de l'organisation des ressources.

4.4.3.4. Gestionnaires d'exécutions d'un workflow [2.14]

Il existe beaucoup de gestionnaires d'exécution de workflows pour les grilles de calcul. A vrai dire, la plupart des grilles actuellement utilisées propose un système permettant de prendre en compte des dépendances entre des travaux à exécuter.

1) Pegasus : est un gestionnaire d'exécution de workflows qui travaille principalement sur la structure du graphe de tâches afin de le réduire pour permettre une exécution efficace. Il utilise donc d'abord un partitionnement du graphe de tâches en sous-groupes de tâches (heuristiques de clustering présentées auparavant). Le gestionnaire place les tâches dans une liste et alloue les calculs prêts aux machines disponibles. Lorsqu'une tâché se termine, ses tâches filles sont à leur tour démarrées.

2) Triana : est un environnement permettant la construction et l'exécution de workflows. Il est développé à l'Université de Cardiff pour le calcul distribué grâces aux services web.

3) GridAnt : est un système de gestion des workflows côté client. Il a été conçu en 2002 à l'Argonne National Laboratory pour les utilisateurs de grilles de calcul dans le but d'être un outil pratique pour exprimer (spécifier des pré-conditions et des tâches exécutables en parallèles) et contrôler les séquences d'exécution. GridAnt ne fournit pas de mécanisme d'allocation automatique des ressources, l'utilisateur doit lui-même spécifier les machines qui exécuteront les travaux contenus dans son workflow.

4.3.4. Heuristique hybride

Sakellariou et Zhao **[2.22]** ont proposé une heuristique hybride pour l'ordonnancement DAG sur des systèmes hétérogènes. L'heuristique combine le mode de dépendance et d'autres modes. L'heuristique calcule d'abord les valeurs du rang de chaque tâche et classe toutes les tâches dans l'ordre décroissant de leurs valeurs de rang. Et puis il crée des groupes de tâches

indépendantes .Dans la phase de regroupement, on traite les tâches dans l'ordre de leurs valeurs de rang et ajouter des tâches dans le groupe actuel. Une fois qu'il trouve une tâche qui a une dépendance à une tâche à l'intérieur du groupe, il crée un autre nouveau groupe. En conséquence, un certain nombre de groupes de tâches indépendantes sont générés. Et le numéro de groupe est attribué sur la base de l'ordre des valeurs de classement de leurs tâches, à savoir, si m> n, la valeur de classement de tâches dans un groupe m est plus élevé que des tâches dans un groupe n. Puis il planifie des groupes de tâches par groupe et utilise un mode batch algorithme pour redéfinir les priorités des tâches au sein du groupe.

5. Conclusion

Dans ce chapitre, on a présenté le problème d'ordonnancement des tâches pour les grilles de calcul, ainsi une vue sur le gestionnaire d'ordonnancement sur grille. On a vu les caractéristiques des ordonnanceurs d'une grille et on a terminé par exposer le problème d'ordonnancement des tâches dépendantes et indépendantes, ainsi que leurs algorithmes d'ordonnancements respectifs, et d'autres heuristiques d'ordonnancement.

1. Introduction [3.1]

Pour améliorer le rendement des grilles, il est important de mettre en œuvre des solutions et des logiciels adaptés au type et fonctionnalité de ces grilles.

Mais il est très difficile de tester ces solutions sur des grilles en pleine production, et les solutions proposées sont sur des architectures indisponibles en réalité, ou en cours de réalisation. Pour remédier à ce problème, divers simulateurs ont étaient développé afin de traiter les comportements des algorithmes d'ordonnancement, réplication et méthodes de transmission des données.

On cite par exemple (NS Networks simulator, GridSim, SimGrid, OptorSim) qui sont utilisés afin de réaliser des simulations, fournissant ainsi un outil d'expérimentation avec un rendement non négligeable.

Les chercheurs ont eu recours à la simulation car ils sont contraint de travailler sur leurs recherches, des plateformes tel EGEE, Grid 5000, Gridpp ne peuvent être accéder pour des expérimentations quand elles sont en plein travail alors on se penche vers les différents simulateurs qui donnes des résultats proches de ceux de tests réels.

2. Simulateur proposé (SimGrid)

2.1. Historique [3.2]

Le projet SimGrid a debuté depuis 1999 en collaboration par Henri Casanova (Universitde Hawaï, Manoa), Arnaud Legrand (Chargé de recherche CNRS, équipe
Mescal de l'INRIA Rhône-Alpes) et Martin Quinson (Maître de Conférences UHP, équipe AlGorille du Loria).ont contribué à l'évolution de SimGrid depuis sa version initiale jusqu'à la version actuelle, chaque amélioration augmente le réalisme des simulations réalisées avec SimGrid et les rapproches de plus vers des modèles réels. SimGrid contient un ensemble de modules chacun destiné à résoudre un certain problème

2.2. Caractéristiques de SimGrid [3.1]

SimGrid est une boîte à outil pour la simulation d'applications distribuées dans des environnements distribués hétérogènes. L'objectif central du projet est de faciliter la recherche dans les domaines de la programmation d'applications parallèles et distribuées, sur des plates-formes distribuées allant du simple réseau d'ordinateurs aux grilles de calcul.

SimGrid regroupe un ensemble de modules, chaque module est destiné à un certain type de simulation: les taches parallèles, transfert de donnés synchrone et asynchrone, simulation des architectures MPI, ainsi que les réseaux pairs à pairs (P2P).

- SimGrid à plusieurs modèles de représentation des ressources par exemples pour le réseau on a CM02, LV08, constant.
- Pour la simulation des réseaux pairs à pairs on peut conduire des simulations hautement évolutives puisqu'on peut simuler jusqu'à 2 mégas d'hôtes sur une même machine.
- SimGrid est offert avec une suite d'applications destiner à faciliter et visualiser des résultats de simulation et des exemples de simulation écrits par les différents utilisateurs de SimGrid et partagé le bias des contributions.
- Chaque ressource (lien de communication ou unit´e de calcul) est associée à une latence (en secondes) et à un débit (en Flop/s pour les ressources de calcul et en O/s pour les ressources de communications).
- On dispose pour chaque ressource de traces (obtenues par mesure sur des plates-formes réelles, indiquant le débit (en Flop/s ou en O/s) disponible à chaque instant sur la ressource et la latence observée à chaque instant.
- Chaque tâche (de calcul ou de communication) est associée à un volume (en Flop ou en O)
- Ainsi, si la tâche T de volume de calcul (ou de communication) S est allouée à l'instant t0 à la ressource R dont la latence est donnée par la fonction L(t) et le débit par la fonction B(t), alors la tâche sera terminée
- (si aucune autre tâche n'est allouée `a cette ressource avant la fin de l'opération) à l'instant T vérifiant

2.3. Vue d'ensemble des composants boîte à outil

SimGrid est développé en langage C, mais il est possible de l'utiliser dans d'autres langages tels que Java ou Ruby, grâce à des API spécifiques. Le code source est donc disponible librement sur le site du projet sous la licence LGPL (Lesser General Public License)

Ce logiciel est découpé en plusieurs modules certains sont particulier tel MSG, GRAS, SMPI, SIMDAG, d'autres jouent les rôles de conteneurs tel SURF et XBT qui gère l'initialisation des simulations et fournissent une plateforme de bases pour les autres modules comme le représente la **figure 3.1**.

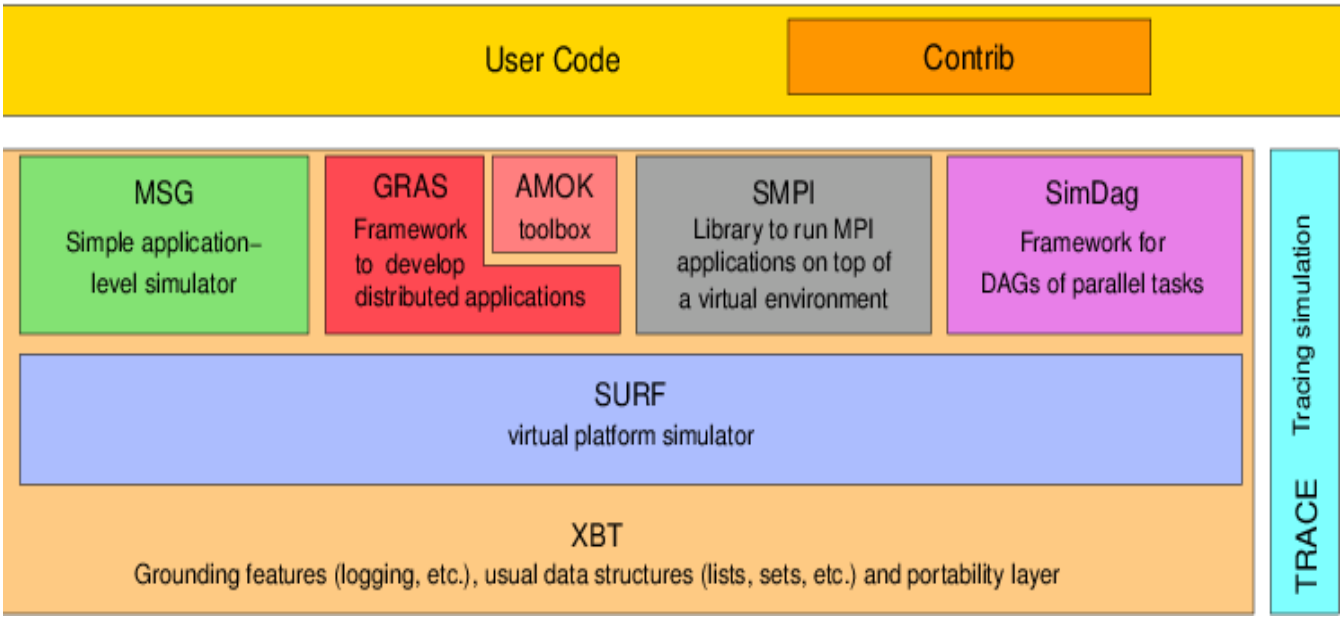


Figure 3.1 : Les différents modules de SimGrid. [3.2]

Les différents modules de simgrid sont partagés en trois couches qui sont :

2.3.1. Couche d'environnement de programmation [3.3]

SimGrid propose plusieurs environnements de programmation construits au-dessus d'un noyau de simulation unique. Chaque environnement vise une audience spécifique et constitue un paradigme différent. Pour choisir lequel d'entre eux on souhaite utiliser, on doit penser à ce qu'on veut faire et ce que serait le résultat de notre travail.

2.3.1.1. Le module MSG (Meta SimGrid) [3.3]

API permettant de simuler des applications distribuées qui ne sont pas destinées à être mises en œuvre dans la réalité et comparé plusieurs heuristique ou algorithme

2.3.1.2. Le module SMPI (Simulated MPI) [3.3]

Il constitue une solution d'émulation pour des codes parallèles. Ce qui aidera à comprendre le comportement d'une application MPI en utilisant des techniques d'émulation.

2.3.1.3. Le module SIMDAG [3.2]

SIMDAG permet de prototyper et de simuler des heuristiques d'ordonnancement pour les applications structurées comme des graphes de tâches (éventuellement en parallèle). Avec cette API, on peut créer des tâches, ajouter des dépendances entre les tâches, récupérer des informations sur la plateforme, ordonnancer des tâches pour l'exécution sur des ressources particulières, et de calculer le temps d'exécution DAG.

2.3.1.4. Le module GRAS (Grid Reality And Simulation) [3.3]

Si on veut développer une vraie application distribuée, alors on peut utiliser GRAS (réalité et simulation de grille). C'est un API pour la réalisation des applications distribuées.

2.3.1.5. Le module AMOK (Advanced Metacomputing Overlay Kit) [3.3]

C'est un module construit à base de GRAS, il est conçu à fonctionner en utilisant des fonctions issus de GRAS mais destiner à être programmé à part comme module.

AMOK permet entre autres de tester la largeur de bande entre deux nœuds. Ce module permet de rechercher la largeur de bande entre des serveurs arbitraires ainsi que la gestion des paires de serveurs. Ce module est efficace pour programmer des architectures clients/serveurs centralisées. Son exécution est toujours reliée au module GRAS comme le montre la **figure 3.1.**

2.3.2. Couche de noyau de simulation (SURF) [3.4]

Cette couche est le noyau interne de tous les simulateurs utilisés dans simgrid et pour les différents modules énumérés précédemment. SURF fournit les fonctionnalités de base pour simuler une plate-forme virtuelle. Il est d'une très basse altitude et n'est pas destiné à être utilisé par les utilisateurs finaux, mais plutôt à servir de base pour des simulateurs de niveau supérieur. Son interface ne sont pas gelés (et ne seront probablement jamais), et l'accent mis sur la performance de la structure sur la facilité d'utilisation. Ce module contient les modèles de plates-formes.

2.3.3. Couche de base XBT [3.3]

La base de l'ensemble du kit est constituée par l'XBT (eXtended Bundle of Touls).

Utilisé par tous les autres modules. Il implémente des fonctions de gestions de la mémoire, ainsi que des structures très utiles, comme les tableaux dynamiques, ou les graphes.

2.3.4. Couche user code [3.3]

C'est la couche qui va contenir le code utilisateur et éventuellement des fonctions et outils issus de contribution.

2.3.5.Tracing simulation

Simgrid peut tracer l'utilisation de la ressource (des hôtes et des liens) en utilisant l'une de ses interfaces de programmation (MSG, SimDAG et SMPI).Cela signifie que le tracé va enregistrer la quantité d'énergie utilisée pour chaque hôte et combien de bande passante utilisée pour chaque lien de la plate-forme. L'idée d'utiliser de traçage est de donner aux utilisateurs simgrid la possibilité de classer les tâches de MSG et SimDAG par catégorie.

2.4. Concept fondamental de SimGrid [3.5]

2.4.1. Agent

Un agent est une entité qui prend des décisions sur le déploiement des programmes. Un agent est défini par un code, les données privées, et l'emplacement sur lequel il sera exécuté.

2.4.2. Emplacement (Location)

Un emplacement (un hôte) est l'endroit dans la topologie simulé par SimGrid sur laquelle un agent s'exécute (fonctionne). Ainsi il est défini par une ressource de calcul, un certain nombre de canaux de transmission qui permettent de communiquer avec d'autres agents, et des données privées qui peuvent être accessibles uniquement par des agents au même emplacement.

2.4.3. Tâche (Task)

Une tâche est une activité de l'application simulée et peut être un calcul ou/et un transfert des données, une tache est définie par une quantité de calcul, d'une taille de données à transmettre, et des données privées.

2.4.4 Chemin (Route)

Un chemin est une agglomération des ressources de transmission (liens). Des emplacements sont alors interconnectés par des chemins.

2.4.5 Le canal de transmission (Chanel)

La communication entre les agents est incluse dans l'abstraction du canal. Un canal englobe la notion de port de communications ouvertes par les agents à des emplacements.

Avec ces abstractions, des algorithmes d'ordonnancement avec simgrid doivent toujours être décrits en termes d'agents qui s'exécutent aux emplacements décris dans le fichier déploiement, et interagissent entre eux en envoyant, en recevant, et en traitant des tâches d'application simulées.

L'agent n'a pas accès direct aux chemins, mais peut envoyer une tâche à un autre emplacement en utilisant un canal. En fait, un emplacement peut avoir de nombreuses boîtes aux lettres et un canal est tout simplement un nombre de boîte aux lettres. Donc, l'envoi d'une tache à un emplacement en

utilisant un canal revient à transférer la tâche sur un chemin particulier, en fonction de l'emplacement de l'émetteur et du destinataire, et le mettre dans une boîte aux lettres particulière.

2.5 Modèles utilisés sous SimGrid [3.4]

2.5.1 Modèle de base

Le Temps d'exécution (Temps de Transfert) est égal à la somme de la latence de la route et le quotient de la taille des données à transmettre sur la bande passante de la route

Temps = Lat + Taille/ Bande passante.

La latence est exprimée en secondes, la taille en (Mflops ou Mb), La Bande est exprimée en (Mflops/s ou Mb/s)

2.6. Fonctionnement de SimGrid

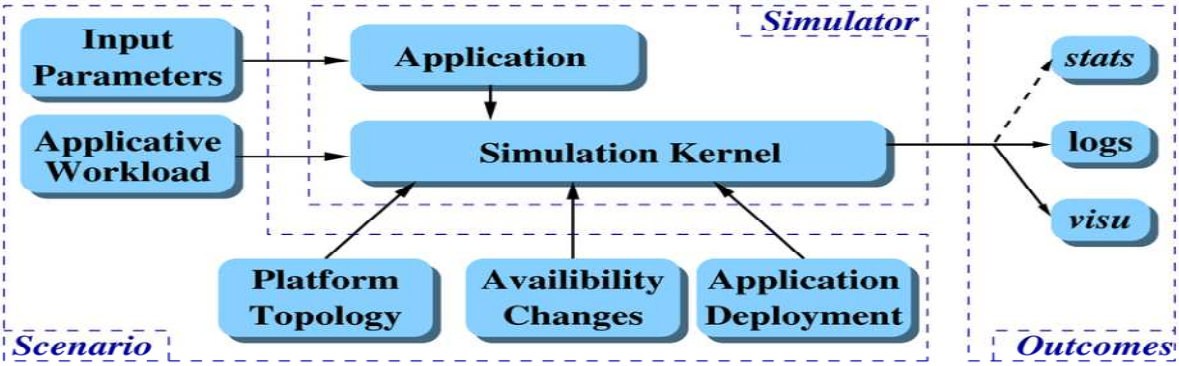


Figure 3.2 : Vue fonctionnelle de SimGrid. [3.4]

Comme le montre la figure 3.2, le noyau réceptionne le fichier XML décrivant la plateforme et celui décrivant le déploiement et d'éventuelles options.

2.6.1. Paramètres d'entrés

En peut considérer plusieurs paramètres d'entrée telle des modèles de réseaux et processeurs stations, en peut les exécuter selon l'exemple suivant :

- --cfg=network/model :{ CM02, LV08, constant, SMPI etc.}
- --cfg=workstation/model :{CLM03, Compound, ptask_107}
- --cfg=CPU/model :{Cas01, Cas01_fullupdate, CpuTI}
- --cfg=network/weight_S :{valeur entière}

2.6.2. Fichier de configuration de charge

Des fichiers de charge peuvent être écrits afin de faire varier la disponibilité et l'état des ressources et leur quantité.

1) *Description de la plateforme :*

Déclaration d'une plateforme nommé « bob » avec une latence= 500000000

```
<platform version="3">
 <AS id="AS0" routing="Full">
    <host id="bob" power="500000000"/>
 </AS>
</platform>
```

2) *Fichier de disponibilté*

Il est également possible de déclarer de façon transparente un hôte dont les changements disponibilité au fil du temps en utilisant l'attribut *Availability_file* qui est un fichier texte distinct dont la syntaxe est illustré ci-dessous a droite.

```
<platform version="3">
 <AS id="AS0" routing="Full">
    <host id="bob" power="500000000" availability_file="bob.trace"/>
 </AS>
</platform>
```

bob.trace

```
PERIODICITY 1.0
0.0 1.0
11.0 0.5
20.0 0.8
```

Au temps 0, notre hôte livrera 500 MFLOP / s. Au moment de 11.0, il fournira la moitié, c'est 250 MFLOP / s jusqu'à l'heure

20.0 où il va commencer à livrer 80% de sa puissance, qui est de 400 MFLOP / s. Enfin, au moment de 21,0 (20,0 plus l'
périodicité 1.0), on boucle du début et l'hôte allons livrer encore 500 MFLOP / s.

3) Fichier d'etat

Pour exprimer le fait qu'un hôte peut changer d'état au fil du temps (comme dans les systèmes P2P), il est possible d'utiliser un fichier décrivant le moment où l'hôte est mise en service ou hors service comme illustré ci-dessous a droite.

```
<platform version="3">
-<AS id="AS0" routing="Full">
   <host id="bob" power="500000000" state_file="bob.fail"/>
  </AS>
</platform>
```

bob.fail ×

```
PERIODICITY 10.0
1.0 -1.0
2.0 1.0
```

Une valeur négative signifie « la machine est hors service » tandis qu'un positif signifie « la machine est active". De temps en temps, 0.0 1.0, l'hôte est allumé. Au moment 1.0, il est éteint et au moment 2.0, il est rallumé jusqu'à l'heure 12 (2.0 ainsi que la périodicité
10.0). Il sera remis en marche au moment de 13,0 jusqu'à 23,0 fois, et ainsi de suite....

2.6.3. Des résultats de traitements

Les résultats de traitement peuvent être des listings de l'affichage de la console, des fichiers traces ou des fichiers écrits par le biais de la console.

2.6.4. La plateforme [3.4]

Elle consiste en un fichier XML contenant la description de la topologie du réseau et station constituant la plateforme, elle est représentée par des stations (host) interconnecter par des routes, qui sont elles aussi décomposables en plusieurs liens.

Chaque lien a les deux caractéristiques fondamentales suivantes :

Une latence : le temps que prend une donnée pour parcourir la totalité du lien.

Une bande passante : représente la taille minimale que peut transférer un lien.

2.6.5. L'application [3.4]

C'est les lancement du fichier déploiement e l'utilisateur, il existe des fonctions permettant de lancer l'application, exemple : Msg_launch_application().

2.7. La construction d'une Simulation Simgrid [3.4]

Un des modules SimGrid : SimDag est utilisé pour construire l'outil de la simulation. Les étapes à suivre dans des programmes SimGrid sont les suivantes :

2.7.1. Description d'application

Définit le code de chaque agent. Plusieurs fonctions existent dans la bibliothéque SimDag pour modeler des agents tels que : SD_task_create, SD_task_get_data, SD_task_set_data.

2.7.2. Description de la plate-forme physique

consiste en un fichier XML contenant la description de la topologie du réseau , les machines et les liens constituant la plate-forme. Lire fichier de la description de la plate-forme peut le faire ou en utilisant une fonction SD_create_environment.

2.7.3. Déploiement d'application

Il consiste en un fichier XML décrivant les noms et les fonctions de chacun des nœuds déclaré dans le fichier plateforme.xml, le nombre de tâches de l'application, la taille de calcul et la taille des communications.

2.7.4. Simulation

La simulation peut etre lancée en utilisant la fonction SD_Simulate.

2.8 Quelques fonctions et type de données utilisées

Cette section décrit les différents types de données fournis par le module SimDag.

Une ressource (station ou liaison) est un endroit où une tâche peut être exécutée. Une station est représentée comme une ressource physique avec des capacités de calcul et a un nom.

- ✓ *SD_workstation_get_list :*

  Retourne la liste de poste de travail.

- ✓ *SD_workstation_get_number:*

  Retourne le nombre de poste de travail.

- ✓ *SD_workstation_get_data :*

  Retourne les données qui utilisent le poste de travail.

- ✓ *SD_route__get_size:*

  Retourne le nombre de liens entre deux postes de travail.

- ✓ *SD_task_create :*

Création d'une nouvelle tâche.

- ✓ *SD_tast_get_data :*

  Retoune les données utilisées par la tâche

- ✓ *SD_tast_get_name :*

Retourne le nom d'une tâche.

- ✓ SD_task_get_start_time:

  Retourne l'heur de début d'une tâche.

- ✓ *SD_task_get_parents:*

Retourne le dynar des parents d'une tâche.

- ✓ *SD_link_get_list :*

Retourne la liste des liens.

- ✓ *SD_link_get_name :*

Retourne le nom de lien.

- ✓ *SD_dependency_add :*

Ajoute une dépendance entre deux tâches.

- ✓ *SD_dependency_remove :*

Suppression d'une dépendance entre deux tâches.

- ✓ *SD_create_environment :*

Création de l'environnent.

- ✓ *SD_get_clock :*

Retoune l'heure actuelle.

- ✓ *SD_daxload :*

Charge un fichier DAX décrivant un DAG (l'application).

- ✓ *SD_exit :*

Détruit toutes les données internes SD.

3. Conclusion

Dans ce chapitre, on a décrit le simulateur SimGrid via son architecture et des différents modules. On découvre un outil puissant et très proche des comportements des environnements en réalité, des différentes plateformes fournissant des résultats efficaces pour la validation des algorithmes et la simulation d'événement de la vie courante.

1. Introduction

Nous avons décrit dans le chapitre 1et 2 les systèmes distribués à grande échelle à savoir les grilles de calcul et le cloud, et expliqué l'un des problèmes majeurs de ces infrastructures, qui est l'ordonnancement des tâches

Les applications candidates à être exécutées sur les systèmes distribués ont des caractéristiques comme : le nombre de tâches, leurs taille de calcul et leur tailles de communications, qui influent sur le temps d'exécution de ces applications.

Dans ce chapitre, nous décrirons en premier lieu les algorithmes Min-Min, HEFT et l'implémentation de HEFT. En deuxième lieu les fichiers indispensables pour la simulation, et en fin les tests comparatifs entre l'algorithme Min-Min et HEFT effectués sous le simulateur SimGrid.

2. Objectif

Après avoir vu un certain nombre d'algorithmes dans le chapitre 2, on a choisit d'étudier Min-min et HEFT et d'implémenter HEFT, et réaliser une étude comparative de ce dernier avec Min-Min sachant que l'implémentation de Min-Min existe déjà (sous SimGrid)

3. Les algorithmes Min-Min et HEFT (Heterogenous Earliest-Finish_Time)

Les algorithmes Min-Min et HEFT sont utilisés dans plusieurs domaine on cite :

Min-min a été utilisé dans : Vgrads et Pegasus

Vgrads : ce projet a élaboré un cadre pour intégrer l'ordonnancement basé sur le suivi des performances des ressources de manière dynamique et évolution constante **[4.1]**.

Pegasus : le projet Pegasus regroupe un ensemble de technologie qui aide les applications à base de workflow pour s'exécuter dans différents d'environnements, y compris les ordinateurs de bureau, les clusters, des grilles, et les Cloud **[4.2]**.

L'algorithme HEFT a été utilisé dans ASKALON qui est un environnement de réalisation d'applications scientifiques de type workflow sous la grille. Le but d'ASKALON est de simplifier le développement et l'optimisation des applications qui peuvent exploiter la puissance de la grille et de cloud computing **[4.3]**.

3.1. Description de l'algorithme Min-Min

L'algorithme Min-min indiqué ci-dessou combine à la fois MET et MCT (vue dans le chapitre 2).

L'heuristique min-min commence avec un ensemble L de toutes les tâches non ordonnancées.

Puis, il calcul le temps d'exécution de chaque tâche sur toutes les ressources disponibles.

Ensuite, la tâche ayant le moins temps d'exécution est sélectionnée (MET) et attribuée à la ressource correspondante (qui va la compléter au plus tôt (MCT)) (d'où le nom de Min-Min). Enfin, la tâche ordonnancée est retiré de L, et le processus se répète jusqu'à ce que toutes les tâches soient ordonnancées**.**

Algorithme Min-Min
Pour toutes les tâches Ti dans la liste des tâches L
    Pour toute machine Mj
        Eij = date_courante + Cij (Cij : temps d'exécution du tâche Ti)
    Fin pour
Fin pour
Tant qu'il existe des tâches non encore ordonnancer
    Pour toutes les tâches Ti avec i = 1..n dans L
        Pour toutes les machines Mj avec j = 1..m
            Trouver le minimum des dates de fin d'exécution Eij
        Fin pour
    Fin pour
    Soit (Ta, Ma) respectivement la tâches et la machine qui correspondent au minimum
    Affecter la tâches Ta à la machines Ma
    Retirer la tâches Ta de la liste L

Mettre à jour les dates de disponibilités des processeurs de machines Ma

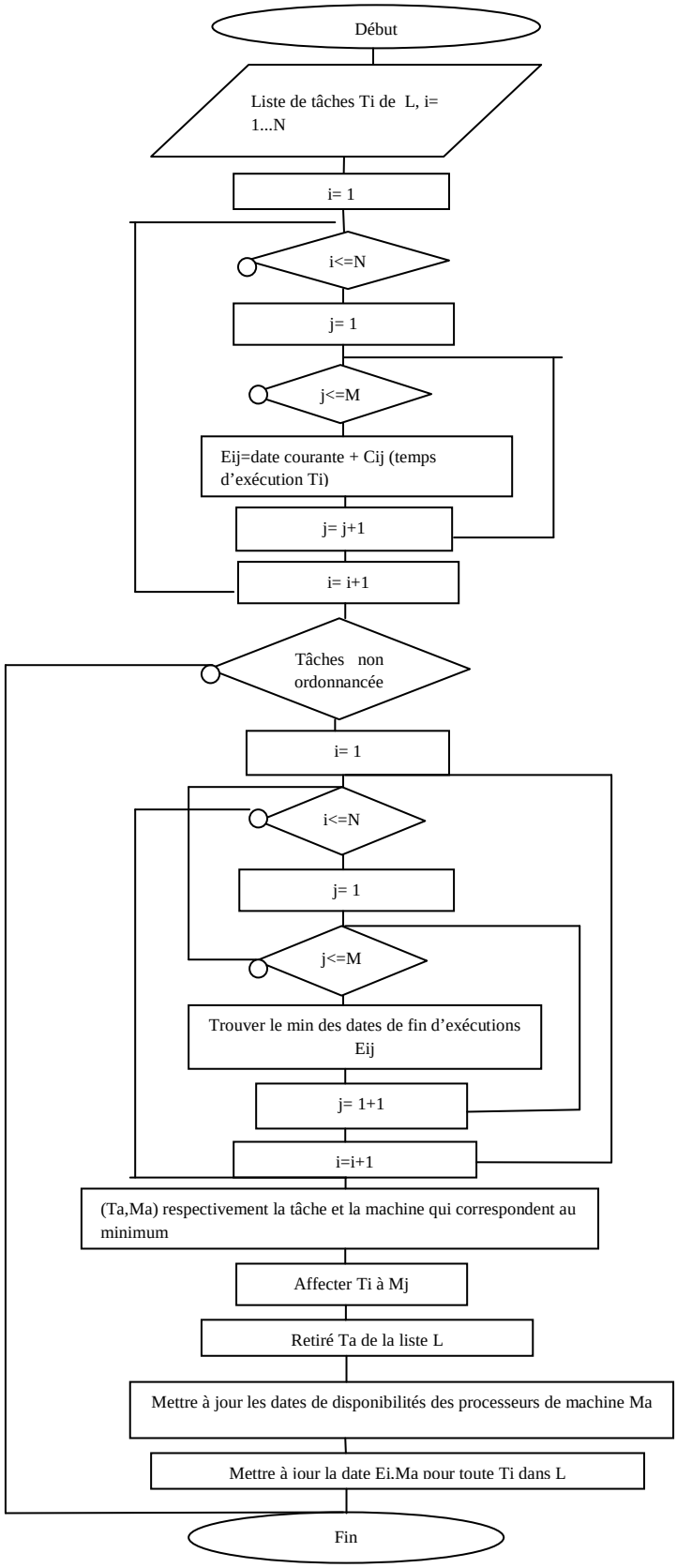
Trier les processeurs de la machines Ma dans l'ordre croissant de leur dates de disponibilités

Mettre à jour la date Ei, Ma pour toute Ti dans L

Fin Tant que

Les étapes de l'algorithme sont décrites dans l'organigramme suivant :

L : une liste de tâches, i : 1..N la liste de tâches, j : 1..M liste de machine.


Début
Liste de tâches Ti de L, i= 1...N
i= 1
i<=N
j= 1
j<=M
Eij=date courante + Cij (temps d'exécution Ti)
j= j+1
i= i+1
Tâches non ordonnancée
i= 1
i<=N
j= 1
j<=M
Trouver le min des dates de fin d'exécutions Eij
j= 1+1
i=i+1
(Ta,Ma) respectivement la tâche et la machine qui correspondent au minimum
Affecter Ti à Mj
Retiré Ta de la liste L
Mettre à jour les dates de disponibilités des processeurs de machine Ma
Mettre à jour la date Ei.Ma pour toute Ti dans L
Fin

La difficulté avec cette heuristique, est d'évaluer le temps de complétion minimum (MCT). En voulant ordonnancer une tâche sur un processeur donné, il faut tenir compte, d'une part des données nécessaires qui se trouve déjà dans l'entrepôt correspondant, et d'autre part des données à transférer à travers le réseau de serveurs. Si des données à transférer sont déjà répliquées dans les entrepôts de plusieurs serveurs, il faut d'abord décider quels serveurs qui serviront de source pour les transferts. Une fois que la source pour chaque donnée transférer a été choisie, il détermine quelles communications doivent avoir lieu.

Au moment d'ordonnancer les communications requises pour une tâche Ti, il faut prendre en compte les communications qui ont déjà été ordonnancées pour les tâches précédentes. Autrement dit, en essayant d'ordonnancer les communications pour une nouvelle tâche, il faut tenir compte du fait que les liens réseau sont déjà occupés à certains moments par des communications précédemment ordonnancées.

L'heuristique Min-Min est séduisant par la qualité de l'ordonnancement produit, mais son coût de calcul est énorme et peut empêcher son utilisation. C'est pourquoi d'autre heuristique ont été conçu (exemple : HEFT) qui sont plus rapides tout en essayant de conserver la qualité de l'ordonnancement produit. L'algorithme *Min-min* est particulièrement coûteux car, chaque fois qu'il tente d'ordonnancer une nouvelle tâche, il considère toutes les tâches restantes et calcule leurs MCT **[4.4]**.

3.2. Description de l'algorithme implémenter HEFT (Heterogneous Earliest-Finish-Time)

HEFT est parmi les algorithmes d'ordonnancements de liste les plus répandus. Il détermine un ordonnancement totalement statique d'un DAG sur un environnement hétérogène de manière à minimiser la durée totale d'exécution de l'application (makespan)

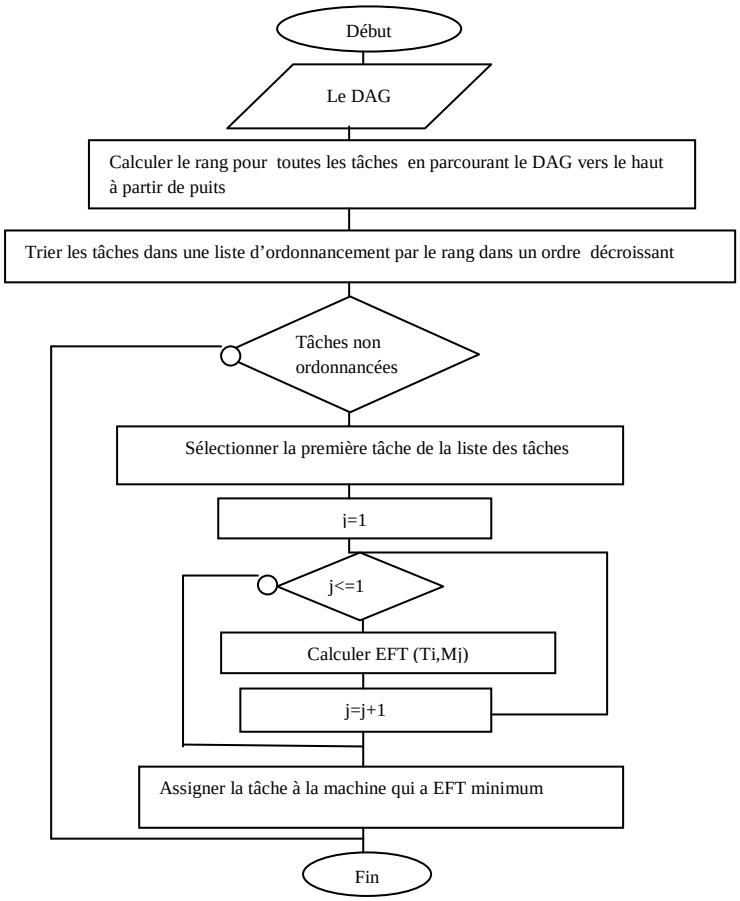
Rang d'une tâche: c'est *la hauteur* (la hauteur d'une tâche est la longueur du chemin le plus long partant de celle-ci vers le puits) ou *la distance maximale* de la tâche par rapport au puits du DAG.

EFT : c'est une fonction qui cherche la ressource qui exécute la tâche le plus tôt possible.

Algorithme HEFT

1. Définir le coût de calcul pour chaque tâche et le coût de communication.
2. Calculer le rang pour toutes les tâches en parcourant le DAG vers le haut à partir de puits.
3. Trier les tâches dans une liste d'ordonnancement par le rang dans un ordre décroissant.
4. **Tant que** y'a des tâches non ordonnées dans la liste
   Sélectionner la première tâche de liste de tâches
5. **Pour** toutes les machines Mj avec j = 1..m
   Calculer EFT (Ti,Mj)
6. Assigner la tâche à la machine Mj qui minimise l'EFT de la tâche
7. **Fin de tant que**

Les étapes de l'algorithme sont décrites dans l'organigramme suivant :



L'algorithme HEFT est largement utilisé comme référence dans les analyses de performances par simulation car les ordonnancements obtenus ont un temps d'exécution raisonnable (**[4.5]**, **[4.6]** **[4.7]**). Lors de l'ordonnancement, la liste de tâches prêtes est triée en fonction de rang des tâches. Plus une tâche à un rang important, plus elle est prioritaire.

L'objectif de cet ordre est d'exécuter les tâches les plus éloignées de puits (nœud ayant aucune arête sortante) du Directed Acyclic Graph (DAG) en premier pour éviter d'importants temps d'inactivité (ressource en attente d'une nouvelle tâche) en fin d'exécution.

Le calcul de l'éloignement d'une tâche par rapport aux puits est effectué en considérant la longueur du chemin le plus long entre la tâche et le puits. La longueur du chemin prend en compte le coût de communications et les temps d'exécutions. Lors de ces calculs, il serait difficile de prendre en compte la congestion du réseau.

Par exemple, lors du calcul du rang d'une tâche, le placement des tâches est complètement inconnu. Sans ce placement, il n'est pas possible de connaitre la quantité de données qui transiteront par un lien réseau et donc de modéliser la congestion du réseau future.

L'algorithme place la tâche sélectionnée sur le processeur minimisant sa date de fin d'exécution (ETF). Le calcul de cette date est réalisé en fonction des transferts de données nécessaires. Le temps de transfert dépend de la congestion du réseau à la date considérée. Apres le placement de cette tâche, l'algorithme peut en ordonnancer une autre juste après, qui nécessitera aussi des transferts de données. Ces derniers risquent eux aussi de modifier les quantités de données transitant par les liens de communications et ainsi perturber les transferts réalisés pour la tâche ordonnancée précédemment **[4.8]**.

La figure 4.1 illustre ce risque de perturbation. Lors du placement de la tâche *C*, la date de début d'exécution de la tâche *A* peut être retardée a cause d'une congestion réseau. Celle-ci est due au transfert des données de la tâche *C* placée après la tâche *A*.

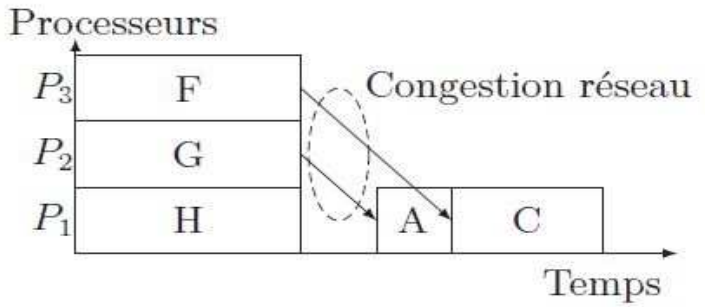


Figure 4.1 : Perturbation sur la date de démarrage d'une tâche placée précédemment [4.8].

4. Le cadre de simulation SimGrid

Pour exécuter la simulation des deux heuristiques Min-min et HEFT, nous avons utilisé le module SimDag de SimGrid 3.8.1.

SimGrid est une boite à outil, qui fournit des fonctionnalités de base pour la simulation d'applications distribuées hétérogènes dans des environnements distribués. L'objectif spécifique de SimGrid est de faciliter la recherche dans le domaine des applications distribuées et parallèles et l'ordonnancement sur les plates-formes de calcul distribué allant de réseau simple de postes de travail aux grilles de calcul et clouds. Le module Simdag est destiné à simulée l'ordonnancement de tâches parallèles organisé comme graphe acyclique orienté (DAG) sur des plates-formes hétérogène.

5. Description de l'environnement de simulation

L'exécution d'une application sous SimGrid nécessite la présence de deux fichiers XML indispensables. L'un décrit la plate-forme cible et l'autre décrira l'application. Le fichier plateforme.xml décrit la plate-forme sur laquelle sera exécutée l'application comme une liste de machines (ressources de calcul ou machines) avec certains paramètres (comme puissance de calcul), ainsi que le réseau reliant ces machines en décrivant les routes (sous forme de liens et leurs caractéristiques comme latence et bande passante). Le fichier application.xml décrit les différentes tâches constituant l'application et la liste des dépendances entre ces tâches et les caractéristiques de ces tâches telles que le temps de calcul et le temps de communication.

6. Description du fichier plateforme.xml

Elle consiste en un fichier XML contenant la description de la topologie du réseau et les machines constituant la plateforme, elle est représentée par des stations (host) interconnecter par des routes, qui sont elles aussi décomposables en plusieurs liens. Chaque lien a les deux caractéristiques fondamentales suivantes :

Une latence : le temps que prend une donnée pour parcourir la totalité du lien.

Une bande passante : représente la taille minimale que peut transférer un lien.

La figure suivante montre un extrait sur la déclaration des hôtes dans le fichier plateforme.xml

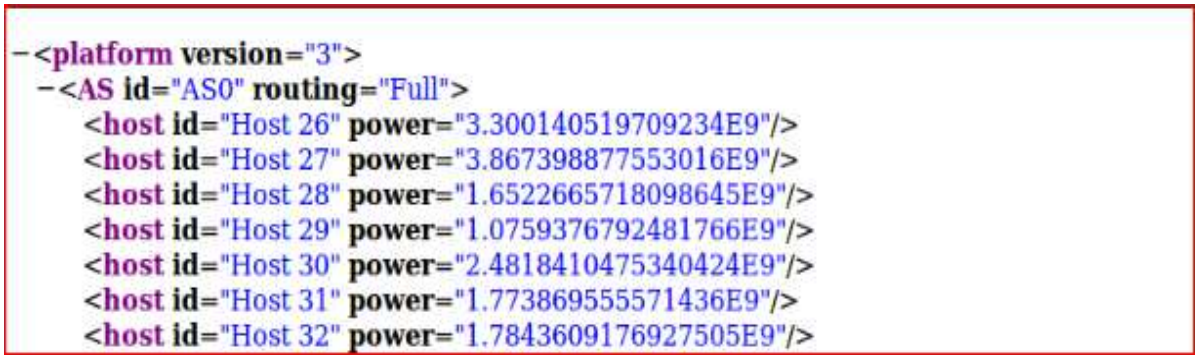


Figure 4.2 : Aperçue sur la déclaration des hôtes dans le fichier platforme.xml

Dans la balise <host> en retrouve les attributs « id » qui correspond au nom du l'hôte et «« power» qui correspond à sa puissance de calcul.

Exemple 1: la première machine est Host 26 ayant une puissance de 3.300140519709234 * e9 flops. La **figure 4.2** correspond à la description des liens réseau de cette plate-forme, la balise <link> contenant le nom du lien représenté par l'attribut « id » ainsi qu'une bande passante représentée par l'attribut « bandwidth » et une latence décrite par « latency ».

Exemple 2: le premier lien et de nom « L152 », sa bande passante est « 1.25*e8 » et sa latence est égale à 1.0*e-4.

La figure suivante montre un extrait sur la déclaration des liens du réseau dans le fichier plateforme.xml

```
<link id="l152" bandwidth="1.25E8" latency="1.0E-4"/>
<link id="l153" bandwidth="1.25E8" latency="1.0E-4"/>
<link id="l154" bandwidth="1.25E8" latency="1.0E-4"/>
<link id="l155" bandwidth="1.25E8" latency="1.0E-4"/>
<link id="l156" bandwidth="1.25E8" latency="1.0E-4"/>
<link id="l157" bandwidth="1.25E8" latency="1.0E-4"/>
<link id="l159" bandwidth="1.25E8" latency="1.0E-4"/>
<link id="l160" bandwidth="1.25E8" latency="1.0E-4"/>
<link id="l161" bandwidth="1.25E8" latency="1.0E-4"/>
<link id="l162" bandwidth="1.25E8" latency="1.0E-4"/>
<link id="l163" bandwidth="1.25E8" latency="1.0E-4"/>
<link id="l164" bandwidth="1.25E8" latency="1.0E-4"/>
<link id="l165" bandwidth="1.25E8" latency="1.0E-4"/>
<link id="l166" bandwidth="1.25E8" latency="1.0E-4"/>
<link id="l167" bandwidth="1.25E8" latency="1.0E-4"/>
<link id="l168" bandwidth="1.25E8" latency="1.0E-4"/>
<link id="l169" bandwidth="1.25E8" latency="1.0E-4"/>
<link id="l170" bandwidth="1.25E8" latency="1.0E-4"/>
<link id="l171" bandwidth="1.25E8" latency="1.0E-4"/>
```

Figure4.3 : Déclaration des liens du réseau.

La figure suivante montre un extrait des définitions des routes entres les machines dans le fichier plateforme.xml

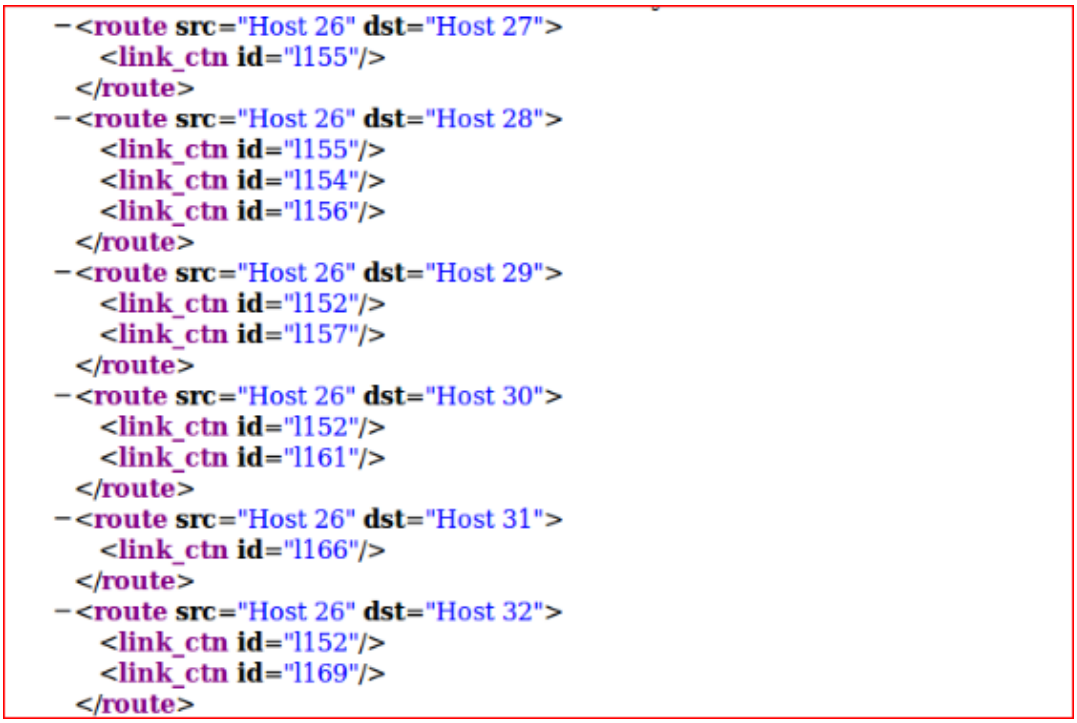


Figure 4.4 : Définitions des routes entres les hôtes.

En ce qui concerne les routes, elles sont définies par les liens les constituant, elles sont décrites par la balise « route ».

Le paramètre « src » pour désigner l'émetteur, « dst » pour désigner le récepteur, et « link_ctn » pour introduire un format de la route.

7. Description du fichier application.xml

Une application de type workflow (DAG) est décrite par 3 parties :

partie1 : liste de tous les fichiers référence (peut être vide) qui sont des entrées/sorties.

partie2 : la définition de toutes les tâches (au moins une), et chacune contient des utilisations qui possèdent des fichiers d'entrée et de sortie qui expriment les dépendances entre ces tâches, comme la montre la **figure 4.5** suivante :

```
<!-- generated: 2008-09-24T14:28:09-07:00 -->
<!-- generated by: shishir [??] -->
-<adag xsi:schemaLocation="http://pegasus.isi.edu/schema/DAX http://pegasus.isi.edu/schema/dax-2.1.xsd" version="2.1" count="1" index="0"
 name="test" jobCount="25" fileCount="0" childCount="20">
 -<!--
    part 1: list of all referenced files (may be empty)
  -->
  <!-- part 2: definition of all jobs (at least one) -->
 -<job id="ID00000" namespace="Montage" name="mProjectPP" version="1.0" runtime="13.39">
    <uses file="region.hdr" link="input" register="true" transfer="true" optional="false" type="data" size="304"/>
    <uses file="2mass-atlas-ID00000s-jID00000.fits" link="input" register="true" transfer="true" optional="false" type="data" size="4222080"/>
    <uses file="p2mass-atlas-ID00000s-jID00000.fits" link="output" register="true" transfer="true" optional="false" type="data" size="4167312"/>
    <uses file="p2mass-atlas-ID00000s-jID00000_area.fits" link="output" register="true" transfer="true" optional="false" type="data"
    size="4167312"/>
  </job>
```

Figure 4.5: la définition de toutes les tâches.

Partie3 : la liste de contrôle de flux de dépendances (parents et fils), comme la montre la **figure 4.6** suivante :

Figure 4.6: la liste de contrôle de flux de dépendances (parents et fils).

Ces applications sont souvent représentées par des workflows. Un workflow (flux de travail) décrit le séquencement des tâches (étapes d'exécution), ces tâches forment un DAG (abordé dans la section suivante) dans lequel les nœuds représentent les tâches et les arêtes représentent les dépendances entre ces tâches.

La figure ci-dessus montre un DAG qui correspond à une application de type workflow :

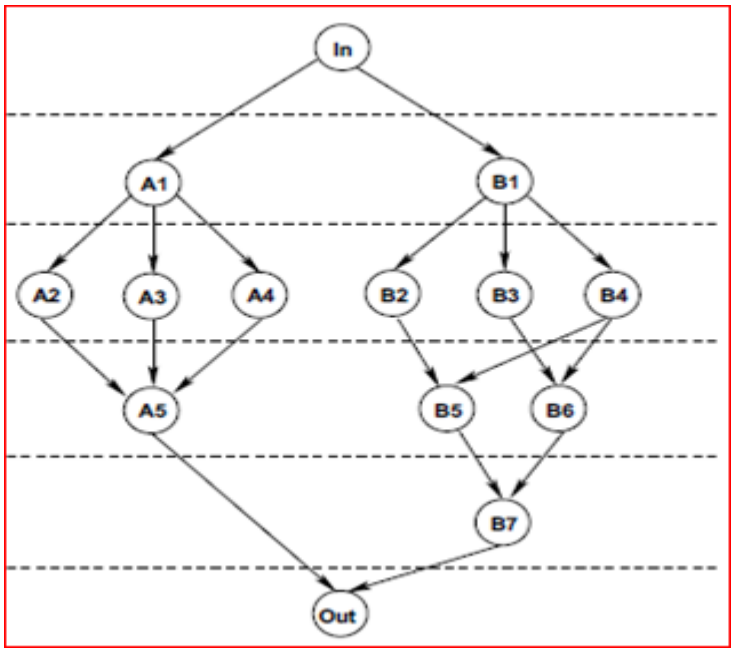


Figure 4.7 : structure d'un DAG d'une application.

Description du DAG : Les nœuds du DAG représentent les tâches à exécuter. L'existence d'un arc (arête) entre deux nœuds représente l'existence d'une contrainte de précédence entre les deux tâches associées aux nœuds. Il est clair qu'il ne peut pas y avoir de cycles dans les relations de précédences sans quoi certaines tâches ne pourraient être exécutées.

Sur ce DAG

La tâche **In** est une source (un nœud n'ayant aucune arête entrante).

La tâche **Out** est un puits (un nœud n'ayant aucune arête sortante).

8. Simulation

Nous avons utilisé différents plates-formes hétérogènes, pour réaliser nos simulations.

Dans l'exemple on a pris une plate-forme qui contient 7 machines et une application de 25 tâches. une application dans le domaine de l'astronomie, dans laquelle les mosaîques du ciel sont créées en fonction des demandes utilisateurs **[4.2]**.

La figure suivante présente le DAG de l'application prise dans l'exemple (application de 25 tâches). La tâche **00** est une source et la tâche **24** est un puits.

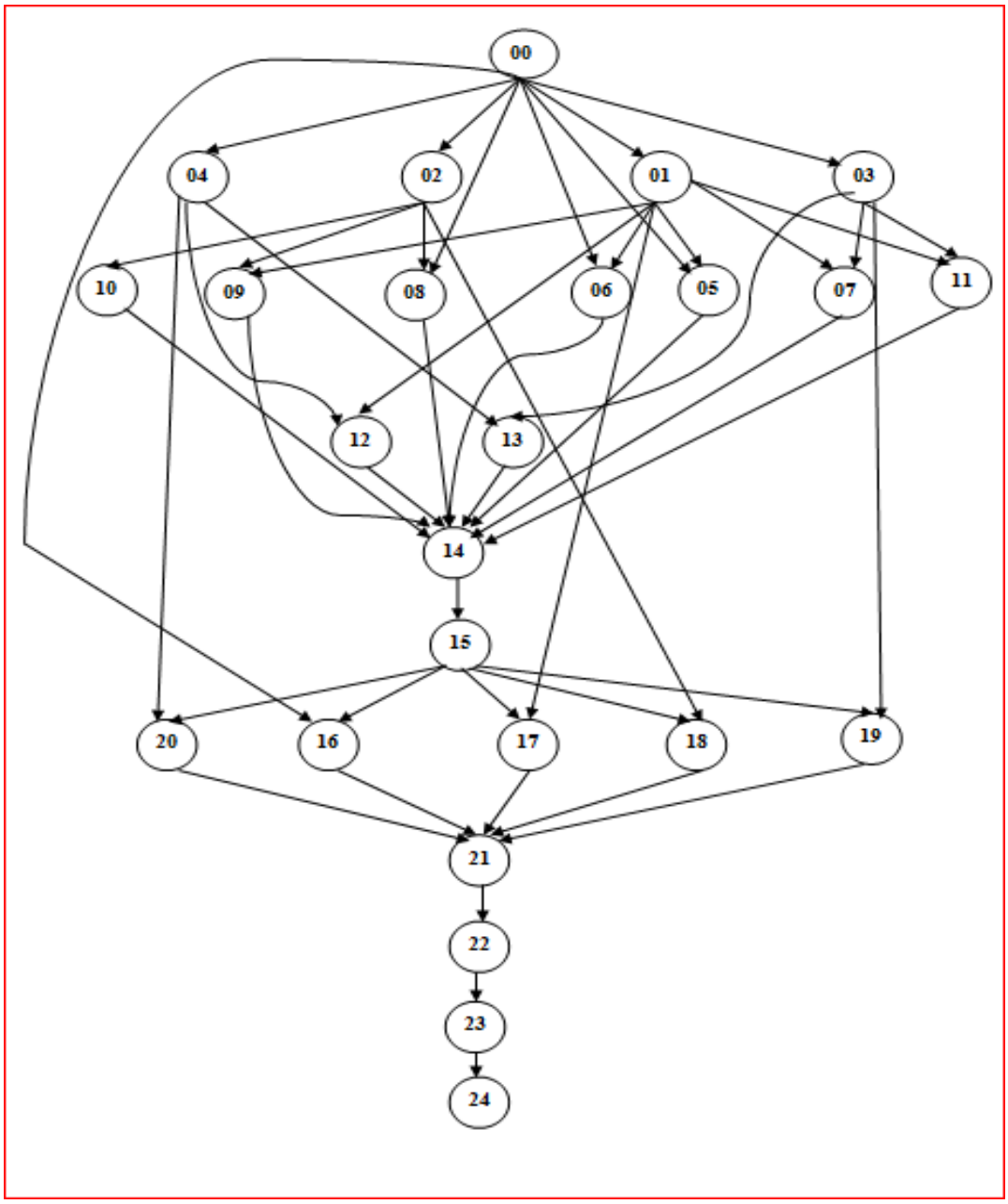


Figure 4.8 : structure d'un DAG de l'application Montage avec 25 tâches [4.2].

La figure suivante donne un aperçu du résultat avec Min-Min :

```
[29.085091] [test/INFO] Schedule ID00010@mDiffFit on Host 27
[29.085194] [test/INFO] Schedule ID00008@mDiffFit on Host 26
[29.989619] [test/INFO] Schedule ID00013@mDiffFit on Host 30
[32.637908] [test/INFO] Schedule ID00009@mDiffFit on Host 27
[32.638011] [test/INFO] Schedule ID00011@mDiffFit on Host 26
[32.638027] [test/INFO] Schedule ID00005@mDiffFit on Host 32
[32.638244] [test/INFO] Schedule ID00006@mDiffFit on Host 31
[32.638351] [test/INFO] Schedule ID00012@mDiffFit on Host 28
[32.638663] [test/INFO] Schedule ID00007@mDiffFit on Host 27
[64.163256] [test/INFO] Schedule ID00014@mConcatFit on Host 27
[64.953548] [test/INFO] Schedule ID00015@mBgModel on Host 27
[66.495779] [test/INFO] Schedule ID00016@mBackground on Host 27
[66.495794] [test/INFO] Schedule ID00017@mBackground on Host 26
[66.495901] [test/INFO] Schedule ID00020@mBackground on Host 30
[66.496001] [test/INFO] Schedule ID00018@mBackground on Host 27
[66.496016] [test/INFO] Schedule ID00019@mBackground on Host 32
[92.289758] [test/INFO] Schedule ID00021@mImgTbl on Host 27
[93.882917] [test/INFO] Schedule ID00022@mAdd on Host 27
[97.173604] [test/INFO] Schedule ID00023@mShrink on Host 27
[101.458976] [test/INFO] Schedule ID00024@mJPEG on Host 27
[101.951429] [test/INFO] Schedule end on Host 27
[101.951856] [test/INFO] Simulation Time: 101.951856
[101.951856] [test/INFO] ------------------- Produce the trace file----------
-------------
[101.951856] [test/INFO] Producing the trace of the run into Montage_25.jed
```

Figure 4.9: Aperçu du résultat avec Min-Min

```
[0.000000] [test/INFO] Schedule ID00002@mProjectPP on Host 27
[0.000105] [test/INFO] Schedule ID00000@mProjectPP on Host 26
[0.000120] [test/INFO] Schedule ID00003@mProjectPP on Host 30
[0.000327] [test/INFO] Schedule ID00004@mProjectPP on Host 27
[0.000434] [test/INFO] Schedule ID00001@mProjectPP on Host 32
[14.576439] [test/INFO] Schedule ID00010@mDiffFit on Host 26
[17.049690] [test/INFO] Schedule ID00008@mDiffFit on Host 27
[29.541537] [test/INFO] Schedule ID00013@mDiffFit on Host 26
[32.637908] [test/INFO] Schedule ID00009@mDiffFit on Host 30
[32.638114] [test/INFO] Schedule ID00011@mDiffFit on Host 27
[32.638217] [test/INFO] Schedule ID00005@mDiffFit on Host 26
[32.638233] [test/INFO] Schedule ID00006@mDiffFit on Host 32
[32.638445] [test/INFO] Schedule ID00012@mDiffFit on Host 31
[32.638548] [test/INFO] Schedule ID00007@mDiffFit on Host 28
[60.428541] [test/INFO] Schedule ID00014@mConcatFit on Host 27
[61.220372] [test/INFO] Schedule ID00015@mBgModel on Host 27
[62.762602] [test/INFO] Schedule ID00016@mBackground on Host 27
[62.762618] [test/INFO] Schedule ID00017@mBackground on Host 26
[62.762724] [test/INFO] Schedule ID00020@mBackground on Host 30
[62.762824] [test/INFO] Schedule ID00018@mBackground on Host 27
[62.762840] [test/INFO] Schedule ID00019@mBackground on Host 32
[88.556582] [test/INFO] Schedule ID00021@mImgTbl on Host 27
[90.149741] [test/INFO] Schedule ID00022@mAdd on Host 27
[93.440428] [test/INFO] Schedule ID00023@mShrink on Host 27
[97.725800] [test/INFO] Schedule ID00024@mJPEG on Host 27
[98.218253] [test/INFO] Schedule end on Host 27
[98.218679] [test/INFO] Simulation Time: 98.218679
[98.218679] [test/INFO] ------------------- Produce the trace file-------------------------
[98.218679] [test/INFO] Producing the trace of the run into Montage_25.jed
```

Figure 4.10 : Aperçu du résultat avec HEFT.

9. Étude comparative

Dans cette partie on étudie les performances des deux algorithmes Min-min et HEFT via une étude comparative.

9.1. Temps de simulation en fonction de nombres d'hôtes de la plateforme

Dans ce cas on a pris l'application de 25 tâches qui s'exécute sur cinq plates-formes différentes contenant respectivement 1, 2, 3,4 et 7 machines.

Nombre d'hosts	Temps de simulation avec Min-Min	Temps de simulation avec HEFT
1	2313,22375	1913,22375
2	5625,65668	4325,65668
3	4584,04039	3884,04039
4	479,762036	339,762036
7	101,951856	98,218679

Tableau 4.1 : Temps de simulation en fonction de nombres d'hôtes de la plateforme

La courbe suivante représente les résultats du **tableau 4.1**:

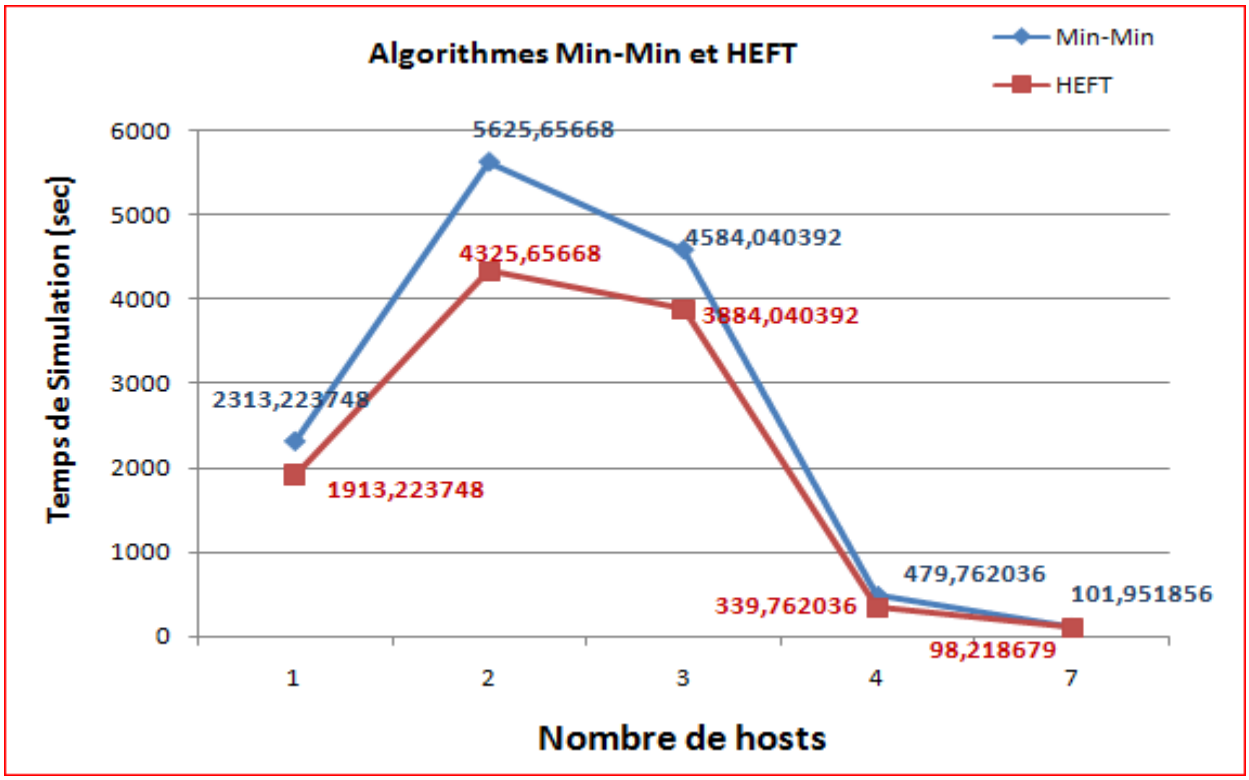


Figure 4.11 : Courbe représentative de temps de simulation en fonction de nombre de hosts.

On relève d'abord de la **figure 4.11** que le temps de simulation de l'algorithme HEFT est toujours meilleur que l'algorithme Min-Min.

En analysant la courbe on constate que :

Le temps de simulation avec 2 machines est supérieur à celui avec une machine.

A noter que pour une machine le temps de simulation est égale au temps de calcul (temps de communication est égale à 0) tandis que avec 2 machines le temps de simulation correspond à la somme du temps de calcul et du temps de communication.

Lorsqu'on a plus de machines (à partir de 3 hosts), les calculs sont distribués sur les machines; ainsi le temps de simulation diminue en augmentant le nombre de machines dans la plate-forme.

9.2. Temps de simulation en fonction de nombre de tâches de l'application

Dans ce cas on a pris une application qui contient respectivement 25 et 50 tâches, s'exécutera sur une plate-forme de trois machines.

Nombre de tâches	Temps de simulation avec Min-Min	Temps de simulation avec HEFT
25	4584,04039	3884,04039
50	9350,06803	8257,06803

Tableau4.2 : Temps de simulation en fonction de nombre de tâches de l'application.

La courbe suivante représente les résultats du **tableau 4.2**:

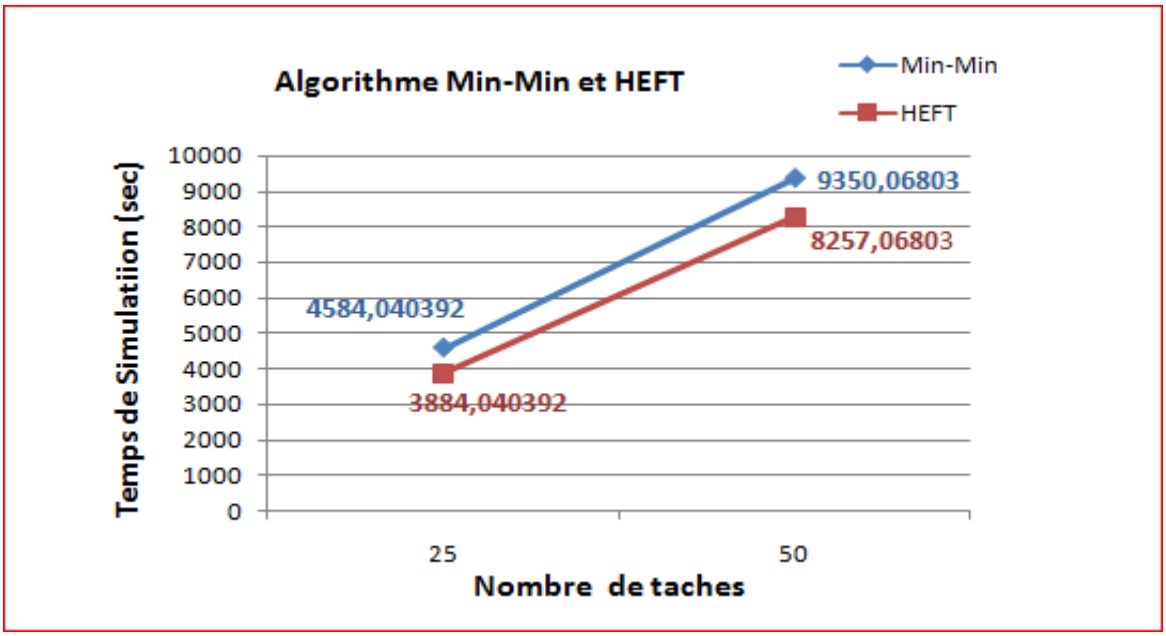


Figure 4.12: Courbe représentative de temps de simulation en fonction de nombre de tâches de l'application.

10. Synthèse

En comparant les différents temps d'exécution des applications sur différentes plates-formes, on constate que les temps de simulation de l'algorithme HEFT sont toujours plus petits par rapport à ceux obtenue avec Min-Min.

11. Conclusion

Après avoir effectué plusieurs tests, il s'est avéré qu'effectivement HEFT minimise le temps de simulation par rapport à Min-Min quelque soit le nombre de machines ou le nombre de tâches.

L'objectif de tout ce travail est de mener une étude sur les algorithmes d'ordonnancement de tâches pour les environnements distribués et hétérogènes, exemple la grille de calcul, afin de les comparer.

Nous nous sommes intéressées au problème d'ordonnancement de tâches sur la grille de calcul. Nous avons présenté une classification de ces algorithmes ; ensuite nous avons détaillé ceux concernant les tâches indépendantes et ceux des tâches dépendantes.

Notre étude s'est intéressée à deux algorithmes qualifiés d'intéressant dans la littérature, à savoir l'algorithme Mim-min et l'algorithme HEFT. Nous avons réalisé plusieurs simulations suivant la nature de la plate-forme utilisée (nombre de machines la constituant) et l'application (nombre de tâches) pour pouvoir comparer ces deux algorithmes sur l'aspect temps d'exécution globale de l'application (makespan) qui est l'un des bons indicateurs des performances des algorithmes d'ordonnancement.

Les études qu'on a menées dans ce travail montrent que le temps de simulation diminue avec l'algorithme HEFT quelque soit le nombre de machines ou le nombre de tâches utilisées.

Ce travail nous a permis de nous façonner une idée sur les plates-formes à grande échelle, et l'ordonnancement des tâches sur une grille de calcul et d'avoir un aperçu global sur leur fonctionnement via l'utilisation de SimGrid.

Perspective

- Une perspective intéressante à mener suite à ce travail serait d'améliorer HEFT pour pouvoir exécuter les tâches indépendantes en parallèle
- Concevoir un algorithme hybride, basé sur HEFT, et qui regroupe les tâches ayant beaucoup de communication entre elles (approche de clustering) et les affecter (assigner) à une même machine, afin d'optimiser les transferts de données entre ces tâches.