ACADÉMIE DE MONTPELLIER

UNIVERSITÉ MONTPELLIER II

— SCIENCES ET TECHNIQUES DU LANGUEDOC —

MÉMOIRE DE STAGE DE MASTER

SPÉCIALITÉ : **Recherche en Informatique**
Mention : **Informatique, Mathématiques, Statistiques**

effectué au LIRMM/INFO

—

sous la direction de MELLE BAERT ANNE-ELISABETH ET MR BOUDET VINCENT

Protocoles optimaux pour le temps de réponse dans les systèmes de calculs à grande échelle

par

Paniez Matthieu

Dates de soutenance : 11-12-13 Juillet 2006

Remerciements

Ce travail, fruit de six mois, n'aurait certainement jamais vu le jour sans l'aide, le soutien et le dévouement de nombreuses personnes.
Je tiens vivement à remercier toutes celles qui m'ont orienté et corrigé. Je pense en particulier à mes tuteurs, Melle Baert Anne-Elisabeth et Mr Boudet Vincent, ainsi que Mr Jean-Marie Alain qui a su me prêter une écoute et du temps.
Je remercie également les personnes du projet PEERATES pour les échanges que l'on a pu avoir.

Table des matières

1 Etude bibliographique **1**

1.1 Qu'est-ce qu'une grille ? 1

1.1.1 Introduction 1

1.1.2 Pourquoi les grids ? 1

1.1.3 Exemple 2

1.2 Définition du problème 3

1.3 Les Tâches Malléables 4

1.3.1 Présentation 4

1.3.2 Définition 4

1.3.3 Le facteur d'inefficacité 5

1.3.4 Compléxité et résolution du problème 5

1.4 Les Files d'Attente 9

1.4.1 Présentation 9

1.4.2 Principales solutions 9

1.4.3 Principaux modèles 11

1.4.4 Optimisations 13

2 Le projet PEERATES **15**

2.1 Présentation du projet 15

2.2 Parallèle avec mon sujet 16

2.3 Détail d'un téléchargement 17

2.4 Le simulateur 18

2.5 Le contrôleur 18

2.6 Les documents 19

2.7 L'échéancier 19

2.8 Les clients 20

2.8.1 Leur création 20

2.8.2 Lecture du plan de téléchargement 21

2.9 Création d'un plan de téléchargement 21

2.10 Négociation d'un transfert de blocs 21

3 Algorithmes et résultats **23**
3.1 Génération de la date d'une requête 23
3.2 Création du plan de téléchargement 24
3.3 Distribution des blocs 24
3.4 Taille des blocs 25
3.5 Les mesures 26

A Organisation d'une classe **28**

B Organisation des classes **30**

Table des figures

1.1 Exemple d'une architecture 2
1.2 Illustration du problème 3
1.3 Correspondance entre graphe de tâches et ordonnancement 5
1.4 Strip-packing : boîte et liste de rectangles 7
1.5 Strip-packing : Exemple et défauts de l'algorithme 7
1.6 Schéma d'un réseau général à K files d'attente 9
1.7 Schéma d'un réseau poissonnien 10
1.8 Exemple de modèle Fork-Join (a) : l'application b) : l'équivalent réseau) 12

2.1 Schéma du réseau PEERATES 16
2.2 Principe d'un téléchargement 17
2.3 Déroulement du temps 20

B.1 Diagramme UML du simulateur 31

Liste des tableaux

1.1 Evolution des performances des processeurs et des réseaux 2

3.1 Valeur de Δt en fonction du processus d'arrivé choisi 23
3.2 Statistiques d'une simulation « classique » 26

Résumé

Dans le cadre de notre Master 2 Recherche, nous effectuons un stage d'environ six mois au sein d'un laboratoire de recherche.
En ce qui me concerne, j'ai choisi le LIRMM avec le sujet de Melle Baert Anne-Elisabeth et de Mr Boudet Vincent sur les protocoles optimaux pour le temps de réponse dans les systèmes de calculs à grande échelle.
Ce mémoire a pour but de présenter ce que j'ai réalisé tout au long de cette période.

Après un résumé des articles que j'ai lus, je présenterai le simulateur du projet PEERATES que j'ai en grande partie réalisé, pour finir sur les algorithmes les plus intéressants et les résultats.

Chapitre 1

Etude bibliographique

1.1 Qu'est-ce qu'une grille ?

1.1.1 Introduction

Depuis 20 ans, devant la demande de calcul de certaines nouvelles applications (la simulation de tremblements de terre, de recherches de signaux extra-terrestre, ...), la première solution était de construire des ordinateurs spécialisés mais très coûteux.
Avec le développement des communications, des puissances des performances globales des ordinateurs (suivant la loi de Moore[1]) et l'émergence d'un système Open Source, une nouvelle solution se met alors en place : utiliser des ordinateurs standards et les relier ensemble pour former une « unité » de calculs : la grille (ou grid).
Avec la démocratisation de l'internet haut-débit, cette notion de grille devient planétaire. De ce nouveau concept apparaîssent de nouveaux problèmes, notamment ceux des ressources, des données et de la communication.

Une grid est une infrastructure virtuelle constituée d'un ensemble coordonné de ressources informatiques potentiellement partagées, distribuées, hétérogènes, externalisées et sans administration centralisée.

1.1.2 Pourquoi les grids ?

Des chercheurs ont commencé à exploiter les ressources de plusieurs PC inutilisés et se sont aperçus que coupler tous ces éléments leur faisait gagner du temps et de l'argent par rapport à l'achat d'un super-calculateur. On pourrait se demander pourquoi les technologies des grilles arrivent soudainement dans notre environnement actuel. La réponse est très simple : elle fait référence à la loi de Moore qui montre que les réseaux

[1] en moyenne, la puissance des processeurs doublerait tous les 19 mois et la vitesse des réseaux tous les 9 mois

évoluent bien plus vite que les solutions de stockages ou la puissance des processeurs.

	1986 à 2000	**2001 à 2010 (prévisions)**
Processeurs	x 500	x 60
Réseaux	x 340 000	x 4 000

TAB. 1.1 – Evolution des performances des processeurs et des réseaux

1.1.3 Exemple

Cette architecture correspond, pour la plupart, à des projets de calculs globaux bien connus. Pendant l'exécution, les « workers » entrent en contact avec le serveur pour obtenir les travaux. Dans la réponse, le serveur envoie un ensemble de paramètres et il peut également envoyer une application, si elle n'est pas déjà stockée dans les « workers ». Quand les « workers » finissent leur travail, ils entrent en contact avec le collecteur pour envoyer les résultats (client). Ici, le client et le serveur sont le même ordinateur.

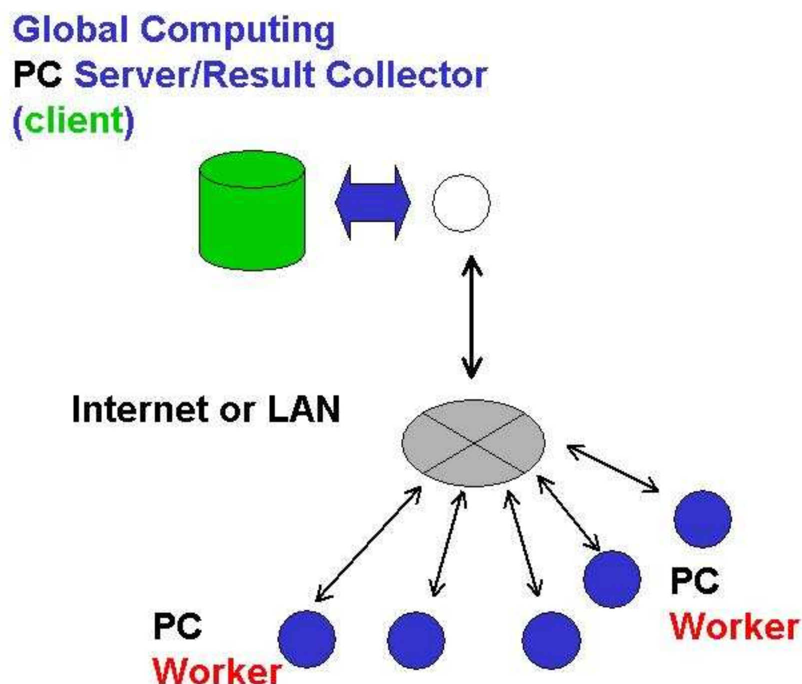

FIG. 1.1 – Exemple d'une architecture

1.2 Définition du problème

Une grille peut-être représenté par un graphe complet (tous les ordinateurs se connaissent) où chacun possède sa propre capacité de calcul (noté c) et est susceptible d'effectuer le calcul ou de redistribuer celui-ci et sont séparés par une distance d. Soit une charge de calcul ω est envoyé sur un noeud qui peut faire 3 choses :
- effectuer le calcul en $\frac{\omega}{c}$;
- donner le calcul à un de ces voisins fait en $d + \frac{\omega}{c}$;
- distribuer ce calcul en plusieurs sous-tâches : $\Sigma\, d_i + \frac{\omega_i}{c_i}$;
- une solution médiane passant par une distribution partielle des calculs et faire l'autre partie du calcul.

A l'heure actuelle, la principale méthode consiste à inonder le réseau pour savoir si des ressources sont disponibles, mais cela occupe inutilement des communications et du calcul.

De ce fait, on s'intéresse à différentes méthodes pour avoir un système de résolution performant de la distribution de calcul.

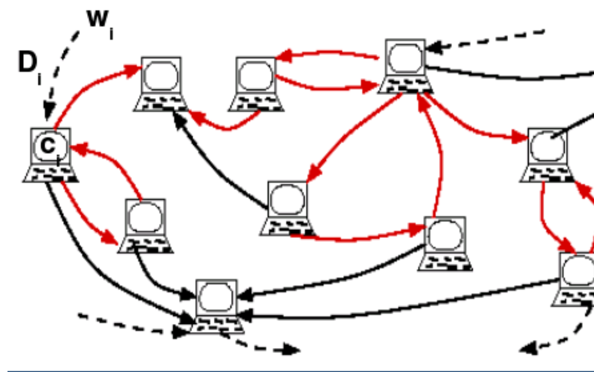

FIG. 1.2 – Illustration du problème

Pour exposer les problèmes liés au placement des données, leur distribution et les communications au sein d'une grille, nous allons voir, dans un premier temps, les tâches malléables.

1.3 Les Tâches Malléables

1.3.1 Présentation

L'un des problèmes majeurs dans la réalisation d'une application parallèle efficace est le choix, pour chaque calcul, du lieu et de la date de son exécution sur la grille ; ce qui revient à résoudre un problème d'ordonnancement.
Dans le modèle standard de distribution de tâches sur une grille de calculs, la gestion des communications rend ce problème souvent plus difficile à résoudre. De plus, elles sont souvent exprimées à l'aide d'un modèle simplifié de l'architecture sous-jacente.
Un nouveau modèle d'exécution a été proposé sur le système de tâches parallèles, dont le modèle des tâches est issu.
Une tâche malléable est le regroupement d'un ensemble de tâches séquentielles exécutées en parallèle par un nombre de processeurs choisi par la stratégie de l'ordonnancement (nombre entier compris entre 1 et n[2]).
Cette représentation permet de concevoir une applicatin parallèle efficace pour plusieurs raisons :
- elles simplifient l'expression du problème car on ne manipule pas explicitement les communications lors du calcul d'ordonnancement ;
- on peut utiliser le même formalisme à plusieurs niveaux de granulité ;
- on peut tenir compte de l'architecture (comme l'hétérogénéité des processeurs).

1.3.2 Définition

Soit $G(V,E)$ un graphe orienté, ou V représente l'ensemble des tâches malléable et E l'ensemble des contraintes de précédence. Soit n, le nombre de processeurs de la machine cible.

Un ordonnancement malléable σ d'un graphe $G(V,E)$ est une paire de fonctions $(Date, Alloc)$, avec $Date\ : V \rightarrow R^+$ et $Alloc\ : V \rightarrow$ 1, 2, ..., n qui assignent à chaque tâche, respectivement, sa date de début d'exécution et le nombre de processeurs alloués.
Le temps d'exécution de la tâche i, distribuée sur $q = Alloc(i)$ processeurs, est noté $t_{i,q}$ et se finit à $Term(i) = Date(i) + t_{i,q}$. Le travail d'une tâche sur q processeurs est $W_{i,q} = q.t_{i,q}$.

[2]si n est le nombre de processeurs disponibles

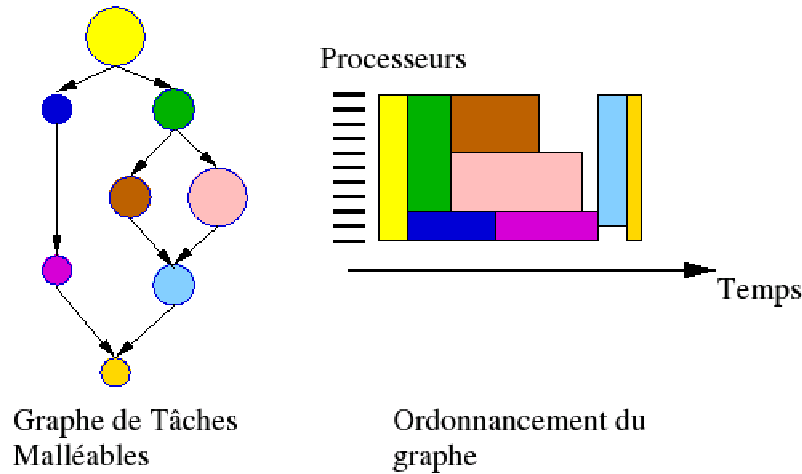

FIG. 1.3 – Correspondance entre graphe de tâches et ordonnancement

Un ordonnancement $\sigma = (Date, Alloc)$ est validé s'il respecte les contraintes suivantes :

contrainte de précédence : $\forall j \in succ(i), Term(i) \leq Date(j)$;

à tout instant, n processeurs sont impliqués : $\forall j \in V, \sum_{\forall i, \substack{Date(i) \leq Date(j) \\ Term(i) > Date(j)}} Alloc(i) \leq n.$

Le temps d'exécution ω d'un ordonnancement σ est la plus grande date de terminaison :

$$\omega = max_{i \in V} Term(i)$$

1.3.3 Le facteur d'inefficacité

Afin de tenir compte implicitement du coût de communication et de synchronisation à l'intérieur d'une tâche malléable, on met en place un facteur d'inefficacité $\mu_{i,q}$, défini par $\mu_{i,q} = \frac{q.t_{i,q}}{t_{i,1}}$. Celui-ci est une fonction croissante du nombre de processeurs exécutant la tâche (coût de mise en place du parallélisme, ajout de processeurs) dont la pente s'adoucit rapidement pour devenir de type exponentiel (les processeurs sont tellement nombreux qu'ils s'attendent mutuellement).
L'estimation de ce facteur d'inefficacité peut passer par une mesure empirique ou par une évaluation analytique de la compléxite des opérations.

1.3.4 Compléxité et résolution du problème

Le problème d'ordonnancement malléable est NP-Complet pour $n \geq 2$ et devient NP-Complet au sens fort si $n \geq 5$ ou si on rajoute des contraintes de précédence pour $n \geq 2$.

La résolution de ce problème peut se traduire par la résolution des deux problèmes suivants :

- un problème d'allocation : le choix du nombre de processeurs $Alloc(i)$ qui vont exécuter la tâche i ;
- la réalisation de l'ordonnancement des tâches définies par $Alloc(i)$.

La plupart des algorithmes actuels traitent ces deux problèmes séparément. La qualité de l'ordonnancement obtenu dépend donc de la qualité des deux algorithmes individuellement et de leur combinaison.
On peut alors employer deux stratégies :

on choisit deux algorithmes indépendants : Soit A_1 l'algorithme qui fournit une allocation $Alloc_1$ et g_1, la garantie de cet algorithme, tel que

$$max(\sum_{i\in V} W_{i,Alloc_1(i)}, CP_{Alloc_1(i)}) \leq g_1\ min_{Alloc} max(\sum_{i\in V} W_{i,Alloc(i)}, CP_{Alloc})$$

CP_{Alloc} est le chemin critique induit par l'allocation $Alloc$.
Soit A_2 l'algorithme d'ordonnancement de graphe multiprocesseurs, de garantie g_2, tel que le makespan ω_2 verifie :

$$\omega_2 \leq g_2\ min(\sum_{i\in V} W_{i,Alloc(i)}, CP_{Alloc})$$

La combinaison de ces deux algorithmes permet d'obtenir la garantie suivante :

$$\omega_2 \leq g_1\ g_2\ min\ max(\sum_{i\in V} W_{i,Alloc(i)}, CP_{Alloc})$$

L'ordonnancement malléable obtenu est de garantie $g_1 g_2$ par rapport a un ordonnancement malléable optimal;

on choisit deux algorithmes imbriqués : L'allocation des tâches malléables permet un ordonnancement multiprocesseur simple (cette approche n'apparait pas dans la partie de mon article).

Nous allon nous intéresser à des algorithmes d'ordonnancement de tâche malléable indépendante, sans relation de précédence, utilisable en pratique, de complexité raisonnable.

Choix d'une allocation

Soit A_{pack} le coût de l'algorithme utilisé pour le placement de m tâches multiprocesseurs indépendantes. En utilisant celui proposé par Turek, Wolf et Yu[2], le strip-packing, la méthode devient polynomiale en $O(nmA_{pack})$, ou n est le nombre de processeurs.
Cet algorithme a été amélioré par Ludwig[3, 4] en $O(nm + A_{pack})$ où la garantie de A_{pack} est fonction de la longueur de la plus grande tâche et la somme des surfaces des tâches.

En ce qui concerne les tâches monotones, on alloue un processeur à chaque tâche. Tant que la quantité de travail total à fournir, divisé par le nombre de processeurs, n'est pas plus grande que le temps d'exécution de la plus grande tâche dans l'allocation courante, l'algorithme alloue un processeur de plus à cette tâche.
Si les tâches ne sont pas monotones, l'algorithme général les exclut en se restreignant au sous-ensemble d'allocations monotones

Empilement

Le placement des tâches allouées précédemment utilise l'algorithme de strip-packing. On peut voir les tâches comme des rectangles que l'on veut mettre dans une boîte (la largeur correspondant au nombre de processeurs et la longueur au temps), le but étant de minimiser la taille de cette boîte.
Cet algorithme est du type d'empilement, qui, dans la pratique, n'est pas forcément

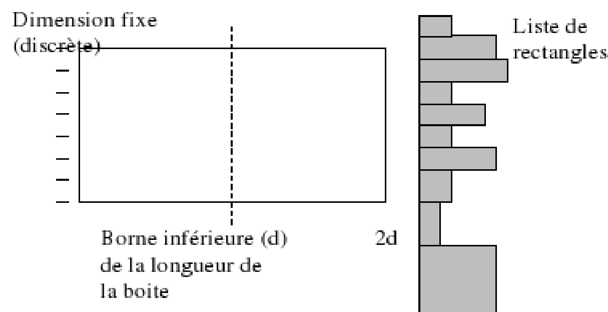

FIG. 1.4 – Strip-packing : boîte et liste de rectangles

adapté. Il applique différentes stratégies de placement suivant la taille de la boîte et celle des rectangles. Si un rectangle est plus long que la moitié d'un côté de la boîte,

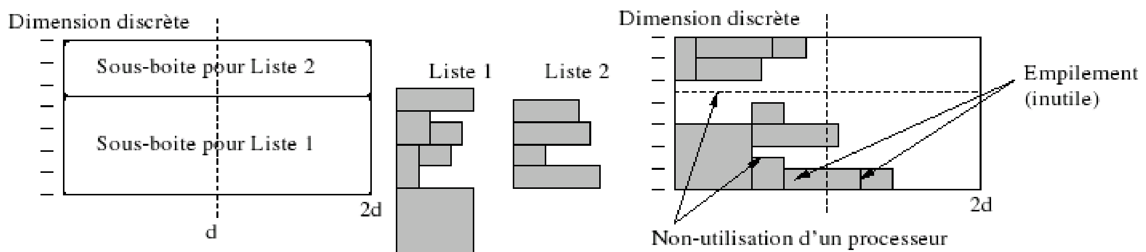

FIG. 1.5 – Strip-packing : Exemple et défauts de l'algorithme

il est placé contre ce côté, et on continue avec l'espace restant. En revanche, s'il existe

une tâche telle que les autres représentent moins d'un quart de la surface de la boîte, elle est placée contre le bord de sa plus grande dimension. Si une paire de rectangles dont chacun des côtés fait plus d'un quart de la dimension de la boîte, ils sont empilés contre le bord.
Ainsi, on va couper la boîte en deux sous-boîtes et la liste en deux de façon à ce que chaque sous-liste puisse rentrer dans une sous-boîte ; ce qui permet de traiter les cas non triviaux efficacement.
Mais ce genre de division de l'espace pose un gros problème : un processeur peut se retrouver « coupé » en deux (cf 1.5).
On peut ainsi remarquer que l'espace n'est pas rempli de façon optimale (laisse un ou plusieurs processeurs inactifs), et qu'il provoque des empilements inutiles.

Dans un second temps, nous allons voir les files d'attente qui mettent en avant les problèmes dûs au temps de réponse et à la qualité de service.

1.4 Les Files d'Attente

1.4.1 Présentation

Nous allons une vue d'ensemble des différentes méthodes utilisées pour l'exécution de tâches parallèles sur des systèmes répartis.
Nous présenterons, tout d'abord, les principales études faites sur le sujet, puis les principaux modèles qui en découlent et qui sont, pour la plupart, très simple (permettant une analyse exacte), à travers leurs possibilités et leurs limitations ; nous finirons par évoquer certaines optimisations.

1.4.2 Principales solutions

Depuis la publication de J.W. Cohen's [1] en 1969, la recherche a mis l'accent sur l'analyse exacte des modèles avec un serveur et/ou une file d'attente ; ce qui a débouché rapidement sur des applications. Ces modèles plutôt simples de réseaux de files d'attente se sont avérés pouvoir apporter des prévisions tout à fait précises du comportement des systèmes informatiques.
Quelques années après, lors de la mise en application de ces méthodes aux systèmes parallèles et répartis, l'adaptation des résultats n'est possible qu'en faisant des simplifications importantes et ne fonctionnant que dans de rares cas.
Ainsi, il a fallut reconsidérer le problème, ce qui a aboutit aux modèles mathématiques suivants.

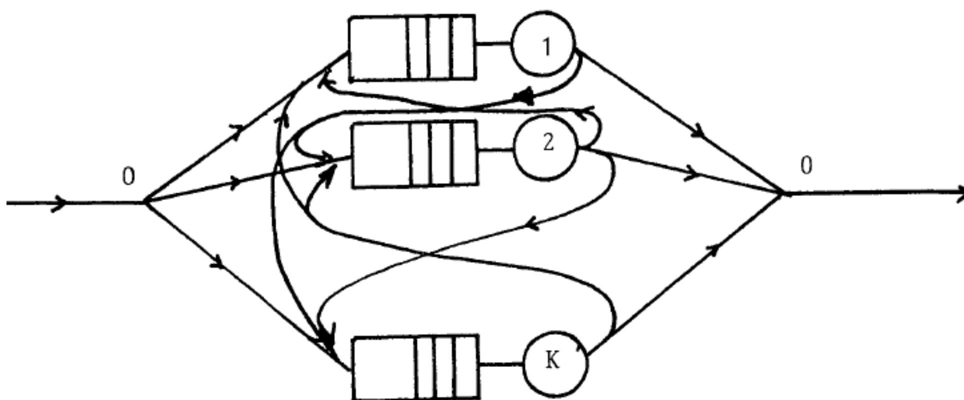

FIG. 1.6 – Schéma d'un réseau général à K files d'attente

La forme produit

Une contribution importante de la théorie de réseau de files d'attente est que, dans certaines prétentions, elle permet d'obtenir une solution exacte et simple pour la distribution commune sous forme séparable : les réseaux de type forme produit.
On considere un réseau de Jackson : les files d'attente sont reliées entre-elles de façon quelconque, la distribution des temps de service étant exponentielle et le processus des arrivées de l'extérieur étant poissonnien. Les files d'attente sont de capacité illimitée pour éviter les blocages et la politique est celle du premier arrivé, premier sorti (FIFO). Ainsi, un client ne peut pas passer deux fois par la même station.

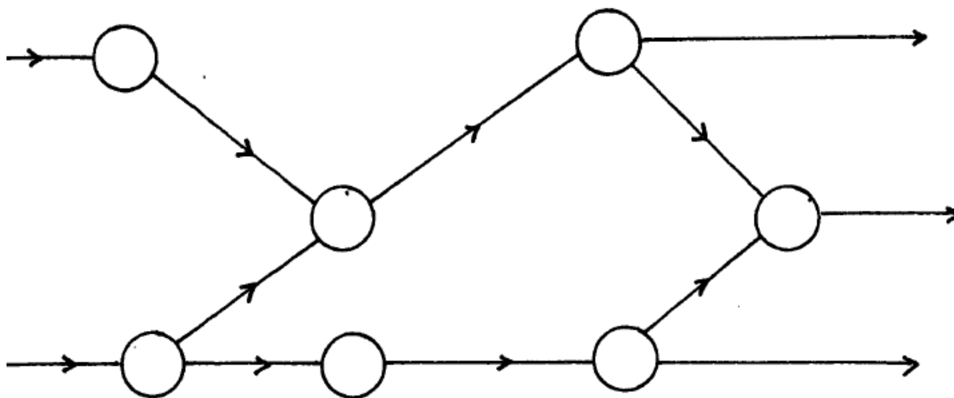

FIG. 1.7 – Schéma d'un réseau poissonnien

La méthode analytique

Elle consiste généralement à exprimer le comportement d'un système sous forme d'un modèle mathématique exact, aboutissant à des formules closes.
Le comportement du système est alors réduit à un ensemble de variables liées entre elles par des équations. L'atout principal de cette approche analytique réside dans le fait qu'elle ne nécessite pas d'outils de support sophistiqués lors des phases de modélisation et de résolution du problème, ce qui convient particulièrement aux phases en amont de la conception d'une application.
Avec cette approche, il suffit d'appliquer la formule pour déterminer les quantités recherchées et surtout pour connaître l'influence relative des différents paramètres. Cette approche, qui requiert couramment de fortes hypothèses (indépendence des événements, conservativité des serveurs, etc.), s'avère praticable pour des systèmes à l'architecture fonctionnelle relativement simple.
En outre, la méthode analytique s'avère bien adaptée à l'étude des «pires cas» de comportement du système, en faisant des approximations simplificatrices et pessimistes.

Cependant, les systèmes logiciels se prettent beaucoup moins bien aux méthodes analytiques, car ils présentent beaucoup de variantes de comportement. En effet, le comportement du système d'exploitation dépend fortement de l'architecture de la machine, sur laquelle on ne dispose souvent que de très peu d'informations. Aussi, les intéractions entre les différents processus peuvent être très complexes et difficiles à caractériser par des lois probabilistes.

La méthode d'approximation lourde et légère

Ces deux techniques d'approximation du trafic présentent une approche alternative aux méthodes numériques pour résoudre les systèmes analytiquement insurmontables. Les techniques d'approximation lourdes et légères du trafic sont parmi les plus populaires.
L'objectif consiste à considérer des situations précises du réseau.
Pour une approximation dite lourde, on se place dans des situations « denses », ou les files d'attente sont pleines, voir en quasi-saturation. Dans le cas d'une approximation légère, on tient compte du taux d'arrivée.
De ce fait, pour une approximation d'un trafic modéré, on réalise une interpolation entre ces deux méthodes.

Equations stochastiques récursives

On suppose qu'à chaque unité de temps, une rafale de clients, avec une taille aléatoire, arrive. Le processus des tailles des rafales est supposé stationnaire et ergodique, et que la distribution des temps de services est de type de Cox.
Le but est d'obtenir explicitement les deux premiers moments du nombre de clients dans la file dans l'état stationnaire. On l'obtient en calculant les deux premiers moments de certaines équations récursives stochastiques satisfaites par notre système. Puis on montre que cette classe d'équations récursives permet d'analyser non seulement une seule file, mais aussi un réseau entier. On étudie enfin le processus de temps d'activité résiduelle de la file $G/G/\infty$ sous des hypothèses de temps d'arrivées et de services stationnaires et ergodiques. On obtient le régime stationnaire unique et montre le couplage à ce régime à partir de tout état initial.

1.4.3 Principaux modèles

Modèle Fork-Join

Soit une classe de réseaux acycliques de files d'attente synchronisées, qui apparaît dans de nombreuses applications parallèles.
Dans un tel réseau, un « Fork » désigne la création simultanée de plusieurs clients déstinés à des files d'attente différentes. Le « Join » correspondant prend place lorsque les services de ces différents clients sont tous terminés.

On établit alors les équations qui gouvernent l'évolution de ces réseaux pour en déduire les conditions de stabilités, ainsi que les bornes inférieures et supérieures sur les temps de réponse. Ces bornes se fondent sur des bornes d'ordonnancement stochastique et sont calculables lorsque les processus d'arrivée et de service sont de renouvellement. Ce modèle est couramment utilisé pour les grâphes de tâches.

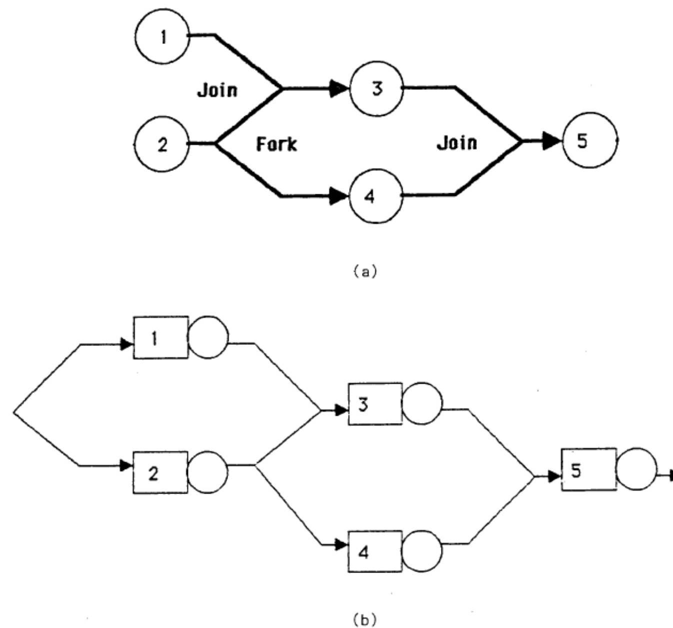

FIG. 1.8 – Exemple de modèle Fork-Join (a) : l'application b) : l'équivalent réseau)

La plus courte file d'attente et la plus petite charge

Il utilise comme base le modèle « Fork-Join ».
Lorsque les clients arrivent, ils se mettent sur la ou les plus petites files d'attente. Le processus d'arrivée est de type poissonnien et celui de service est exponentiel. On essaye ainsi d'équilibrer la charge dans le système.
De ce fait, on limite les calculs au maximum afin de prendre une décicion des plus rapides.

Modèle du vote

Dans un système distribué, afin de résoudre les problèmes liés à une décision ou un choix, on peut ce servir de la notion de vote : n entités communiquantes proposent une valeur, effectuent un calcul, et doivent se mettre d'accord sur une ou plusieurs valeurs communes.
Ce type de problème est largement utilisé dans les systèmes répartis ; par exemple pour

garantir la cohérence des données distribuées sur plusieurs sites.
Il a été montré que ce problème n'a pas de solution dans un système asynchrone, et plusieurs solutions ont été proposées pour contourner ce résultat d'impossibilité. Toutes les solutions proposées jusqu'à maintenant renforcent le système d'exécution, soit en imposant des bornes temporelles, soit en y ajoutant des « boîtes noires » garantissant des propriétés additionnelles, comme les détecteurs de défaillances.

Modèle Time-warp

Cette approche consiste à ne jamais bloquer la progression d'un processus. Elle considère un modèle de simulation comme un ensemble d'objets communiquant par envoi de messages.
Tout objet a une horloge propre qui gère un temps virtuel local. Un objet peut communiquer avec n'importe quel autre objet et il possède une file d'attente de messages en attente d'un traitement.
Dans cette file d'attente, les messages sont triés par ordre croissant des dates. Le traitement d'un message provoque la mise à jour du temps virtuel local de l'objet.
Quand un objet reçoit un message dont la date est inférieure à la date de son horloge virtuelle locale, il applique une procédure de retour arrière qui se décompose en trois phases :

Restauration : il restaure l'état de l'objet à une date antérieure ou égale à la date du message retardataire ;

Annulation : il annule les effets des messages qui ont été exécutés par l'objet et dont la date est postérieure à celle du message retardataire. Cette technique repose sur l'utilisation des anti-messages qui consistent à renverser l'effet des messages ;

Accostage : il intègre les nouveaux changements qui ont eu lieu entre la date du message retardataire et le dernier message exécuté.

1.4.4 Optimisations

Equilibrage de charge

Un aspect opérationnel des systèmes répartis est la disponibilité d'un protocole qui équilibre de façon optimale la charge de travail en amont des serveurs : un protocole d'équilibrage de charge.
On peut distinguer deux catégories :

Modèle de cheminement : dès qu'une requête arrive, elle est acheminée sur un des serveurs du réseau ;

Modèle d'attribution : en fonction de la source, les serveurs déterminent s'ils prennent la requête ou pas.

Un protocole d'équilibrage de charge tient compte également de l'information qu'il a besoin pour fonctionner, qui peut être une connaissance totale du système à celle

restreinte d'un objet particulier.
Ce système peut fonctionner sous des caractéristiques dépendantes du temps et est alors appelé statique ; par opposition, on trouve des systèmes dits dynamiques.

Routage

Quand on fait du routage dynamique de messages qui arrivent aléatoirement, le contrôleur d'un réseau de communication obtient l'information sur l'état de congestion des noeuds en aval qu'après un dèlai considérable ; cette information peut être obsolète aux moments des prises des décisions.
On considère la situation où des tâches arrivent selon un processus de Poisson et elles doivent être envoyées vers une des files d'attente avec un temps de service distribué selon une loi exponentielle (dont la moyenne peut dépendre de la file), sans avoir connaîssance de l'état de la congestion dans l'une des files d'attente. Néanmoins, la distribution de probabilité (conditionelle) des états de la file d'attente qui n'est pas observée, peut être calculée dans le routeur.
Nous dérivons la distribution de probabilité jointe des états des deux files d'attente en fonction de la politique de routage. Ce calcul permet d'identifier des schémas de routage optimal pour deux types de contextes : l'optimisation globale dans laquelle on minimise la somme pondérée des longueurs moyennes des files d'attente, et dans le cadre de l'optimisation individuelle, où le but est de minimiser l'espérance du délai de chaque tâche individuelle.

Allocation de serveur

La durée d'une visite sur un serveur d'une file particulière est aléatoire et ne dépend pas de l'état de la file visitée. Deux variantes sont considérées : soit la file visitée est vidée à la fin de chaque visite, soit seuls les clients présents à l'arrivée du serveur quittent le système à la fin de sa visite. La politique d'ordonnancement détermine la prochaine file d'attente que devra visiter le serveur. Sous l'hypothèse où les processus des arrivées sont homogènes, les résultats suivants sont obtenus : si le contrôleur ne connait pas l'état du système, une politique cyclique minimise, à chaque instant, le nombre moyen de clients. Dans le cas où le contrôleur connait l'état de chaque file d'attente, la politique qui alloue le serveur à la file la plus longue minimise, à chaque instant et pour l'ordre stochastique fort, un vecteur ordonné du nombre de clients. Ce modèle s'applique à certains protocoles de communication multi-accès.

Chapitre 2

Le projet PEERATES

Peu de temps après le début de mon stage, notre groupe de recherche APR a été contacté par une entreprise voulant faire du téléchargement légal, en s'appuyant sur le principe du réseau Peer-to-Peer.

2.1 Présentation du projet

Le projet a pour but de mettre en place une plate-forme accessible à tout le monde, tout en garantissant un débit minimum (un temps maximum).
Afin de garantir le caractère légal et la qualité du service, la présence d'un Central, que l'on appelera « Contrôleur », est indispensable. Celui-ci donne au client authentifié une liste de petits serveurs (les « partenaires ») sur lesquels le client va pouvoir télécharger les différents petits fichiers (« blocs ») qui composent le document demandé (le tout schématisé par la figure 2.1).

Le Contrôleur, qui est unique, est la machine qui pilote la distribution. Il prend les décisions concernant :
- le découpage d'un document en blocs ;
- la duplication et la répartition de ces derniers sur les partenaires ;
- le contrôle de l'authentification du client ;
- la création d'un plan de téléchargement pour le client ;
- la surveillance de l'état du réseau.

Les partenaires, qui possèdent une certaine quantité de « blocs » de documents différents, doivent être utilisés de façon optimale afin de minimiser le coût et la charge du réseau.
Une fois authentifiés, les clients reçoivent un plan de téléchargement et essayent d'optimiser leur bande passante en téléchargeant en parallèle les différents blocs.

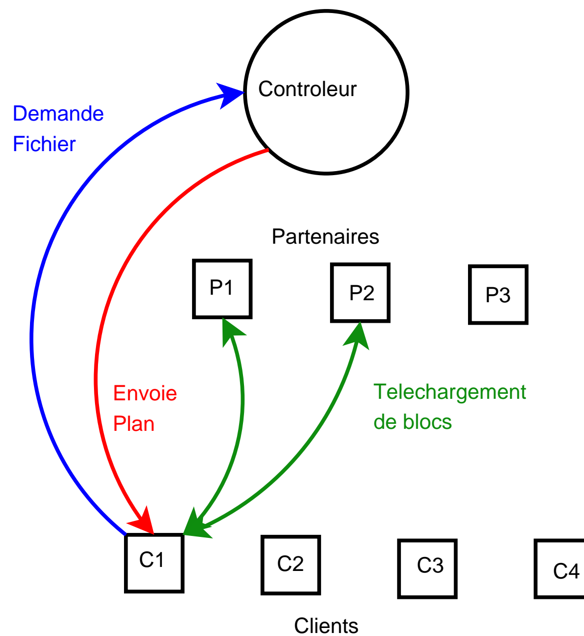

FIG. 2.1 – Schéma du réseau PEERATES

2.2 Parallèle avec mon sujet

Mon sujet d'origine consistait à considérer un graphe représentant un système de calculs à grande échelle (une « grille ») afin d'optimiser le temps de réponse et d'équilibrer la charge par le biais d'un simulateur de charge.
Si on considère que le réseau PEERATES (milliers de partenaires et centaines de milliers de clients selon leur estimation) est la grille de calcul, le temps de traitement est remplacé par le temps de téléchargement et l'équilibrage de charge passe par le calcul et l'optimisation du téléchargement en parallèle, d'un client sur plusieurs partenaires.

À l'heure actuelle, on sait modéliser et réaliser concrètement un téléchargement, mais la problématique provient des volumes. Les réseaux Peer-to-Peer existants (overnet, gnutella et dernièrement torrent) répondent à ce problème mais en posent un autre : la légalité des fichiers qui y transitent.
Le projet PEERATES devrait répondre à ces deux critères. Le simulateur permettrait de fixer certains paramètres (nombre de partenaires, nombre de réplications, etc) et de valider la stabilité du réseau et les cas extrêmes (tels que l'écroulement ou les pics).

2.3 Détail d'un téléchargement

Après la vision globale de la plate-forme, nous allons détailler le mécanisme d'un téléchargement.

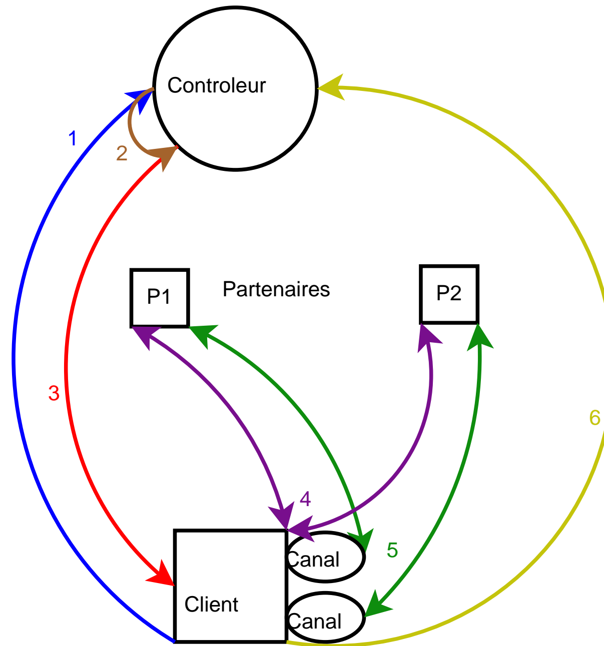

FIG. 2.2 – Principe d'un téléchargement

Dès que le client a choisi le fichier à télécharger, il se connecte au Contrôleur (1) qui l'authentifie. Ce dernier cherche dans sa base de données le fichier (2), crée le plan de téléchargement (dont les différentes stratégies seront vues en 3.2), et le lui envoie (3).
Dès la réception, le client parcourt séquentiellement le plan et, pour chaque bloc, procède à une négociation de débit (4) puis, si le partenaire le peut[1], ouvre un canal qui télécharge le bloc correspondant (5).
Une fois l'ensemble des blocs téléchargés, le client envoie un rapport de qualité au Contrôleur (6), servant à la mise à jour de l'image qu'il a du réseau.

[1]le partenaire peut-être indisponible ou refuser une connexion s'il a déjà un certain nombre de téléchargement

2.4 Le simulateur

Dans un système homogène, on arrive, selon les modèles, à calculer la répartition de la charge. Le problème survient dès que le système devient hétérogène et assez important ; de plus, on rajoute ici l'aspect dynamique (le réseau évolue au cours du temps). De ce fait, on arrive rapidement à des formules très complexes ou à des problèmes NP-Complets.

Comme l'objectif est d'arriver à une optimisation de la bande passante du réseau, j'ai eu la charge de créer un simulateur qui reproduit, le plus fidèlement possible, l'ensemble des opérations du vrai réseau. Ainsi, on arriverait à fixer certains paramètres (de façon optimale) pour les éliminer lors de modélisations mathématiques.

C'est avec cet objectif que mon travail s'est concentré sur trois points (les deux derniers étant assez liés) :
- réalisation du simulateur ;
- optimisation de certains algorithmes ;
- exploitation des résultats.

Le simulateur commence par lire le fichier de configuration (toujours un fichier et des paramètrès par défaut) pour fixer les paramètres fondamentaux.
Puis il met en place les structures fixes (contrôleur, échéancier, partenaires[2]) ; la simulation proprement dite peut alors commencer.

2.5 Le contrôleur

Le contrôleur se crée lors de la construction du simulateur. Il va contenir la liste de tous les documents avec leurs blocs ainsi que tous les partenaires.
Une fois mis en place, le simulateur lui donne le nombre de partenaires avec leurs différentes caractéristiques à ajouter ; mais il ne crée pas les documents immédiatement, sous peine de voir un temps de simulation (bien qu'aucune simulation n'aurait commencé) très important (une liste chainée ayant autant d'éléments que de $nombre\ de\ blocs \times nombre\ de\ documents$). Quand le contrôleur reçoit une demande de fichier, il vérifie s'il le contient, crée le document le cas échéant, et retourne un plan de téléchargement : de ce fait, les documents ne sont créés qu'à la demande (un peu comme les clients), dans le but d'améliorer la vitesse d'exécution.

[2]on a 4 catégories de partenaires, classés par leur débit : 20ko/s, 32ko/s, 64ko/s et 128ko/s

2.6 Les documents

Comme la plate-forme PEERATES va accueillir tous types de fichiers, sa taille est un paramètre important. De plus, un fichier peu connu ne sera pas téléchargé à la même fréquence qu'un document très populaire ; ainsi, la notion de popularité est également importante (probabilité que ce document soit téléchargé).
De ce fait, si on considère T_1 la taille totale d'un document sur le réseau, D la taille d'un bloc, n_b le nombre de blocs et r le nombre de fois qu'un fichier est dupliqué, on a alors :

$$T_1 = D \times n_b \times r.$$

Actuellement, tous les documents sont répliqués r fois. Mais, en fonction de sa popularité, celui-ci va être plus ou moins dupliqués (voir pas du tout[3]) pour pouvoir s'ajuster à la demande. Ainsi, si K est l'espace disque utilisé par un partenaire pour stocker les différents blocs, T_2 la taille totale disponible sur le réseau, n_d le nombre total de documents et P le nombre de partenaires, on a les relations suivantes :

$$T_2 = P \times K$$
$$T_1 \times n_d \leq T_2.$$

Par exemple, si on a 10000 fichiers[4] de 420 Mo[5] avec 3000 partenaires de 5Go, alors la valeur de r a son importance :
- si $r \leq 3$, alors l'espace est suffisant ;
- si $r > 4$, le réseau ne peut pas tout contenir.

Comme je l'ai dit précédemment, la valeur de r n'est pas fixe ; dans l'idéale, elle varie dynamiquement en fonction des demandes faites à tout instant.
Le simulateur permettra de fixer ce paramètre, du moins en apprécier une estimation.

2.7 L'échéancier

Cette partie du simulateur est l'une des pièces maîtresses.
En effet, le simulateur éxécute un certain nombre de tâches qui dépendent d'événements déclencheurs (programmation événementielle) qui, à leur tour, ajoutent de nouveaux événements dans l'échéancier.
D'un point de vue chronologique, l'échéancier commence par contenir l'ajout des partenaires (créés avant même toute simulation), puis la première requête d'un client (qui permet également de « prévoir » les prochains clients), et enfin le début d'un téléchargement avec la date de fin prévue correspondante (schéma chronologique sur la figure 2.3).

Cette date de fin est régulièrement déplacée dans l'échéancier car le débit d'un téléchargement est susceptible de varier au cours du temps.

[3] dans ce cas de figure, il reste sur le contrôleur et le client le téléchargera directement
[4] correspond à peu de fichier comparativement au réseau torrent ou Overnet
[5] correspondant à 60 blocs de 7Mo

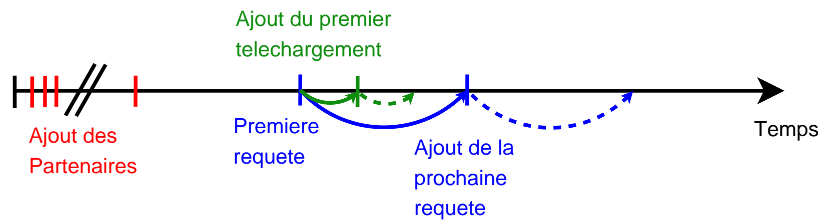

FIG. 2.3 – Déroulement du temps

Etant donné qu'il contient l'ensemble des événements (liste doublement chaînée), passés et futurs, l'échéancier grossit de façon très importante, en fonction du nombre de requêtes (2 fois le nombre de blocs au minimum). Comme on réalise des ajouts continuellement et des suppressions régulièrement, il a fallu trouver un moyen pour réduire le « coût » de ces opérations au maximum.

La premiere optimisation vient de la taille de l'échéancier.
En effet, plus la taille de celui-ci est petite, plus son parcours sera rapide : en conséquence les événements, une fois traités, ne servent plus et sont donc supprimés de l'échéancier.

La seconde provient de l'optimisation de la fonction d'ajout d'un événement.
En fait, il existe deux fonctions d'ajout :
- une, dite classique, qui parcourt l'échéancier depuis le début et insère l'événement à la place adéquate ;
- une seconde qui limite le parcours de l'échéancier.

Cette seconde prend en paramètre le nouvel événement à insérer, mais également, celui qui a provoqué ce nouvel événement. Comme il se situe forcément dans le futur, je me place dans l'échéancier au niveau de l'événement déclencheur et je le parcours depuis cet endroit jusqu'à le mettre à la bonne place. Pour une même date, on trouve d'abord les événements de fin de téléchargement, puis les autres sortes.
Je me sers également de ce modèle pour faire les suppressions.

Ces optimisations ont permis de réduire de façon très conséquente le temps d'une simulation.

2.8 Les clients

2.8.1 Leur création

Au début du stage, les clients (qui correspondent au nombre de requêtes que l'on veut simuler) étaient générés en même temps, juste après les partenaires.

Le problème survient lorsque l'on veut augmenter le nombre de clients : le temps de génération devient trop important et il a fallu trouver une nouvelle méthode.

Une fois la première requête générée, selon l'algorithme d'arrivée choisi (nous les détaillerons en 3.1), je génère une date aléatoire qui va prévoir la date de la future requête. Ainsi, tous les clients sont crées au fur et à mesure, et cette charge de calcul devient « insignifiante » pour le simulateur.

2.8.2 Lecture du plan de téléchargement

Ce plan est lu par le client de façon séquentielle. Il contient une première ligne avec des informations générales (nombre de blocs, identifiant du client) suivie d'autant de lignes que de blocs à télécharger. Chaque colonne correspond respectivement à :

- l'identifiant du partenaire possédant le bloc ;
- l'identifiant du bloc ;
- le statut du téléchargement.

A la fin de la lecture de ce plan, si certains blocs n'ont pu être téléchargés, le client fait une seconde lecture (« passe » dont le nombre dépend du fichier de configuration mais mis à 2 par défaut).
Pendant cette seconde passe, si un partenaire refuse toujours (c.f. 2.10) ou est indisponible, et suivant le nombre de blocs restant, le client demande un nouveau plan de téléchargement ou prend ce(s) dernier(s) bloc(s) sur le contrôleur (on veut bien sûr limité ce processus au maximum).

2.9 Création d'un plan de téléchargement

Cette tâche est faite par le contrôleur.
Il possède une liste de documents avec leurs blocs correspondants et, pour chacun, une liste de partenaires, dont le nombre dépend d'un paramètre prédéfini, sur lesquels les blocs se trouvent. A noter que les partenaires ont également une liste des blocs.

Selon l'algorithme choisi (que nous verrons en 3.2), le contrôleur sélectionne un partenaire de la liste pour chacun des blocs du document, puis renvoie ce plan de téléchargement au client correspondant.

2.10 Négociation d'un transfert de blocs

Cette négociation a pour objectif de mettre en relation un client et un partenaire pour le téléchargement physique d'un bloc, mais surtout de choisir un débit optimal pour l'un et l'autre.
Il faut, dans un premier temps, que le partenaire soit disponible.

Une fois cette condition remplie, le client propose un débit au partenaire (le client a un débit physique limité à 256ko/s[6], mais que l'on diminue volontairement à 128ko/s[7] pour qu'il puisse faire autre chose sur internet). Le partenaire, selon son débit physique et le nombre de ses connections (il ne peut pas proposer un débit inférieur à 12ko/s[8] sauf si le client force cette connection), fait une estimation de ce qu'il peut proposer. On a ainsi plusieurs cas :

- le partenaire n'a pas de connection en cours et peut proposer toute sa bande passante ;
- le partenaire a déja des connections mais n'a pas rempli toute sa bande passante : il propose le restant et regarde s'il peut diminuer les autres connections pour en proposer davantage ;
- le partenaire n'a plus de bande passante et regarde si, en équilibrant uniformément cette nouvelle connection, il peut l'accepter.

Cette estimation faite, il renvoie la plus petite des deux (entre la sienne et celle du client) ou un refus de connection (si le débit proposé est inférieur à 12ko/s).
Le client, s'il reçoit un débit, peut ouvrir un canal au débit convenu, et répéter l'opération autant de fois que sa bande passante le lui permet. Dans le cas d'un refus, si le client doit télécharger son dernier bloc, il force et recommence une négociation (le partenaire peut alors allouer exceptionnellement un débit inférieur à 12ko/s) ; sinon il passe au bloc suivant et traitera ce bloc lors de la prochaine passe.

---

[6]en kilo-octets par secondes correspondant à 2Mbits/s
[7]correspondant à 1Mbits/s
[8]correspondant à 96kbits/s

Chapitre 3

Algorithmes et résultats

3.1 Génération de la date d'une requête

Dans le fichier de configuration, une option permet de sélectionner le type d'algorithme pour cette génération avec le taux d'arrivée (en requête par heure).
En effet, on peut choisir entre une arrivée de type :
- périodique ;
- poisson ;
- uniforme.

Pour ces algorithmes, on va considérer A le taux d'arrivée et Δt le temps entre deux requêtes. La fonction $\Omega(x,y)$ est un tirage aléatoire uniforme[1] en format double entre

Type de processus	**Valeur de Δt**
Périodique	$\frac{1}{A}$
Poisson	$\frac{(-\log\ (\Omega(0,1)))}{A}$
Uniforme	$\Omega(0,\frac{2}{A})$

TAB. 3.1 – Valeur de Δt en fonction du processus d'arrivé choisi

x et y.

Il reste à implémenter les processus OnOff et MMPP.

[1] algorithme de génération uniforme fourni par Mr Boudet Vincent

3.2 Création du plan de téléchargement

Actuellement, il existe deux algorithmes :
- aléatoire uniforme ;
- aléatoire uniforme pondéré.

Pour un bloc donné, le contrôleur a le choix d'autant de partenaires différents que du nombre de réplications mis dans le fichier de configuration.
La première méthode consiste à tirer aléatoirement un partenaire de la liste.
Pour la seconde, et étant donné que les partenaires n'ont pas les même débits, on somme les débits de ceux ayant le bloc, puis on choisit de facon aléatoire un nombre entre 0 et cette somme. A celle-ci, on soustrait, au fur et à mesure, le débit de chacun (dans l'ordre décroissant) et on obtient le partenaire.
Par exemple, on a 3 partenaires (respectivement P1, P2 et P3) de 20ko/s, 64ko/s et 128ko/s, soit un total de 212ko/s. Ainsi, vu que la distribution est uniforme, on a respectivement 9,43% de chance de prendre P1, 30,19% P2 et 60,38% P3.
De cette manière, on essaye de favoriser au maximum les partenaires ayant les plus gros débits.
Après quelques simulations, j'ai constaté que ce modèle apportait le résultat escompté, mais pas de façon « convenable ». J'ai donc essayé de modifier un peu ce dernier algorithme pour favoriser davantage les partenaires ayant les plus gros débits : au lieu de prendre juste la somme, je lui retranche la moitié du plus petit débit. Dans l'exemple précédent, on obtient alors 4,95% pour P1, 31,68% pour P2 et 63,37% pour P3. De ce fait, les plus petits débits[2] sont très peu sollicités et on arrive, dans leur cas, à des situations quasiment optimales : ils ne refusent presque plus de connections.
En revanche, les partenaires ayant les plus gros débits (vu qu'ils sont nettement plus favorisés) commencent à en refuser.
Il reste donc à trouver le bon compromis grâce à plus de simulations.

3.3 Distribution des blocs

Cette distribution se fait, dans le simulateur et de façon implicite lors de la création du plan de téléchargement.
Elle consiste à répartir, selon l'algorithme sélectionné, les blocs sur les différents partenaires. Etant donné que les blocs sont dupliqués un certain nombre de fois, ce paramètre influence directement la distribution.
Cette distribution peut se faire de façon périodique ou aléatoire uniforme. Dans le premier cas, il suffit prendre un partenaire P_1 qui va nous servir de référence et, selon une certaine fréquence f tirée aléatoire, place les blocs sur tous les partenaires $P_1 + f$, pour tous les blocs.

[2]les partenaires ayant 20ko/s et 32Ko/s comme débits

Pour le moment, le simulateur utilise la distribution selon la loi uniforme, car la distribution périodique ne tient pas compte du nombre de partenaires de chaque classe, et par conséquent, les différents débits. En effet, si f est trop petit ou trop grand, la distribution va se « cantonner » aux partenaires ayant des petits débits et ralentir ainsi les téléchargements correspondants.

3.4 Taille des blocs

Ce point me semble assez important dans ce genre de réseau.
En effet, le réseau torrent, le plus performant à l'heure actuelle, utilise des blocs de 250ko. Etant donné que nous sommes parti sur une base de 7Mo, la différence me semble énorme.
L'objectif d'un petit bloc est de limiter au maximum le nombre de téléchargements simultanés sur un partenaire pour qu'il puisse éviter les refus au maximum. Cet atout se fait au détriment du nombre de blocs en cours de téléchargement sur tout le réseau (1 pour 28) et au surcoût du nombre de négociation de débits qu'il engendre.
Sur les simulations que j'ai pu réalisées, le temps de simulation dépend essentiellement du nombre de blocs qui transitent dans le réseau, mais je n'ai pas encore pu mettre en évidence une comparaison entre ces deux choix (ou un autre intermédiaire).

3.5 Les mesures

Nos simulations contiennent deux types de mesure :
- mesures on-line qui sont calculées à chaque événement d'un client ou d'un partenaire ;
- mesures off-line ou post-simulation, calculées après une simulation et qui concernent le réseau dans sa globalité.

Je me suis attaché à réalisé la deuxième partie.
Ainsi, si on considère une simulation classique (1000 partenaires, 1500 clients, des fichiers de 140Mo et les différents algorithmes mis en aléatoire uniforme), on obtient :
On peut voir que la répartition de la charge n'est pas efficace, qu'on est dans une

Débit d'un partenaire	**20 ko/s**	**32 ko/s**	**64 ko/s**	**128 ko/s**
Nombre de partenaires	126	250	500	124
Nombre de downloads max. simultanés	190	591	2075	374
Ratio $\frac{ligne3}{ligne2}$	1.51	2.36	4.15	3.02
Nombre de connections refusées	1542	1932	451	0
Nombre de blocs téléchargés	2205	5650	14384	3711
Nombre de blocs forcés	78	118	23	0

TAB. 3.2 – Statistiques d'une simulation « classique »

situation de quasi-saturation en ce qui concerne les partenaires des deux premières classes, alors que la charge globale du réseau n'est que de 20%.

Conclusion

Avant tout, ce stage m'a permis de réunir le côté théorique et pratique de l'informatique dans le domaine des réseaux.
En effet, la compréhension des articles m'a donné une vision ouverte sur les différentes méthodes déjà étudiées. J'ai pu en tirer profit lors de la conception du simulateur ou certains algorithmes qui le composent.
La réalisation d'un simulateur était quelque chose de nouveau, tout comme la programmation événementielle. L'exploitation optimale du simulateur pourra, dans très peu de temps, fournir la plupart des réponses restées jusqu'ici en suspens.

Annexe A

Organisation d'une classe

Dans le but d'avoir un projet le plus lisible, modifiable et évolutif possible par le maximum de personnes, j'essaye de respecter les mêmes constructions et programmations sur l'ensemble du projet.

L'ensemble de mes classes, mises à part celles qui contiennent le « main », sont construites avec un fichier code source et un fichier qui sert de librairie (*i.e.* : « bloc.h » et « bloc.cpp »).

Le fichier qui me sert de librairie est construit de la façon suivante :
- inclusion des librairies extérieures nécessaires au fonctionnement de cette classe ;
- attribut(s) et/ou méthode(s) privé(s) de la classe ;
- attribut(s) et/ou méthode(s) public(s) de la classe.

Exemple avec « bloc.h » :

```
using namespace std;
#ifndef BLOC_H
#define BLOC_H

#include ...

class Bloc
{
  private :
                    ...

  public:
                    ...
};
#endif
```

En ce qui concerne les fichiers sources, ils respectent la structure suivante :
- inclusion de la librairie correspondante ;
- le constructeur ;
- le destructeur ;
- la méthode qui permet d'afficher les différentes caractéristiques de la classe ;
- les autres fonctions du programme ;
- les accesseurs et mutateurs correspondant à l' (aux) attribut(s) privé(s).

Exemple avec « bloc.cpp » :

```
#include "bloc.h"
//Constructeur
Bloc::Bloc(int id,int id_doc)
{
        ...
}


    //Fonction d'affichage
    void Bloc::affich_Bloc()
{
    ...
}

    //Autre(s) fonction(s)
    ...

    //Accesseurs et mutateurs
    void Bloc::SetId(int nb)
{
      Id=nb;
}


    int Bloc::GetId()
{
      return Id;
}
    ...
```

De cette manière, on peut se repérer facilement au sein du code source.

Annexe B

Organisation des classes

A la première approche, je pensais que le simulateur contiendrait le contrôleur qui lui-même contiendrait l'ensemble de mes classes.
Lorsque j'ai voulu mettre en place des clients, à travers la classe « Client », j'ai eu un problème de dépendances de mes classes (dépendance cyclique). De ce fait, j'ai décidé que les clients appartiendraient à ma classe simulateur et non au contrôleur, ce qui semble le plus logique à l'heure actuelle.
De plus, lorsqu'il a fallu intégrer l'échéancier, je suis arrivé à la même conclusion.
De ce fait, j'ai ma classe « Peerate », qui n'est en fait que le « main », qui contient le simulateur. A son tour, elle possède les classes « Client », « Echéancier », « random » et « Contrôleur ».
L'échéancier contient les événements ; le client possède la classe « FileAttente », car, dans un soucis d'évolutivité, je considère qu'un client peut avoir autant de « canaux » que l'on veut.
Le contrôleur possède la classe « Partenaire » et la classe « Document », qui contient elle-même la classe « Bloc ».
Le tout est détaillé sur la figure B.1.

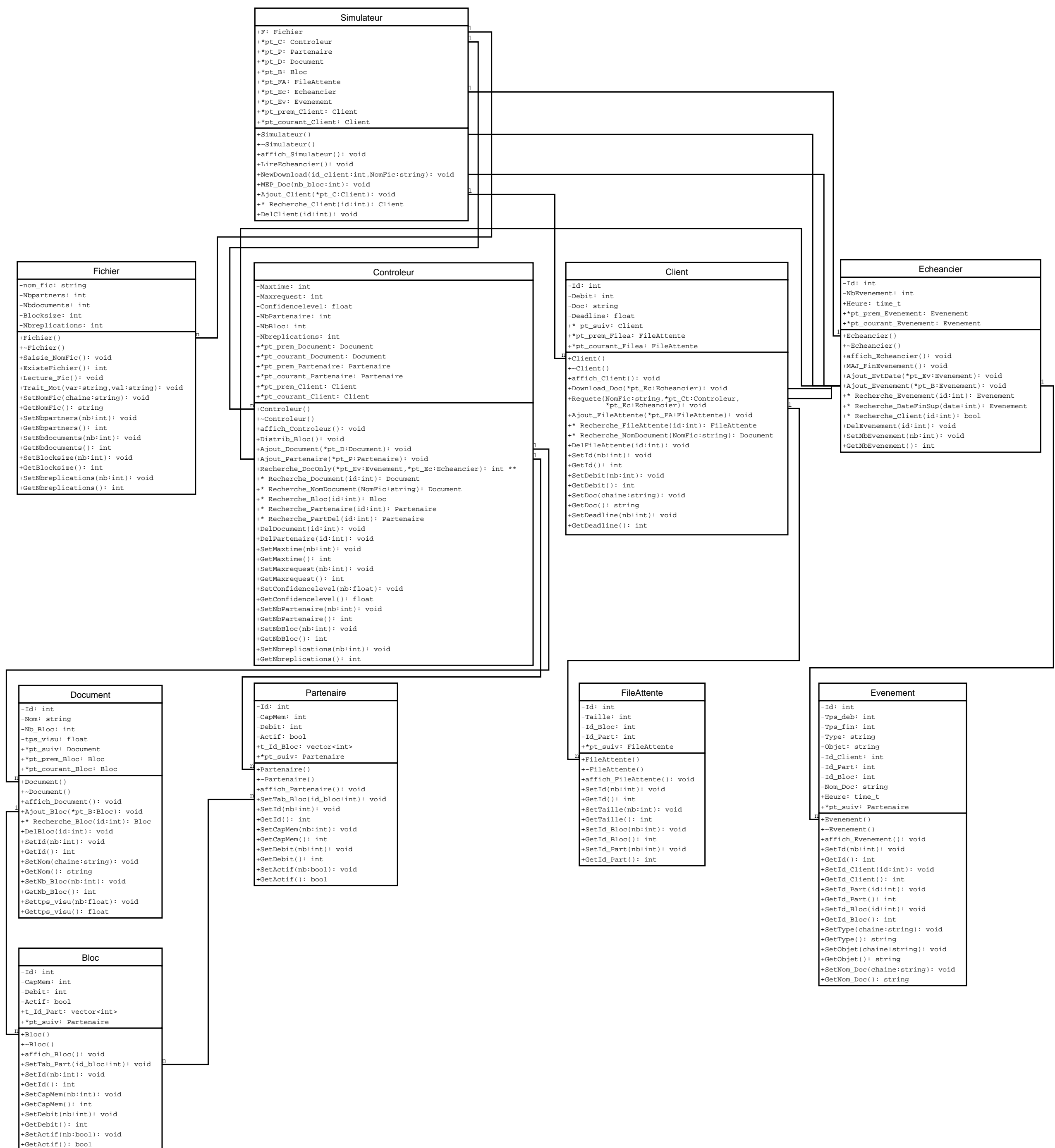

FIG. B.1 – Diagramme UML du simulateur

Bibliographie

[1] J.W. Cohen, *The Single Server Queue*, North-Holland - Amsterdam, 1982.

[2] J. Turek, J. Wolf, and P. Yu., *Approximate algorithms for scheduling parallelizable tasks. In th Annual ACM Symposium on Parallel Algorithms and Architectures*, Addison-Wesley, 1992.

[3] W. T. Ludwig., *Algorithms for scheduling malleable and nonmalleable pa- rallel tasks. Technical Report CS-TR-95-1279*, Madison, August 1995.

[4] W. T. Ludwig and P. Tiwari., *Scheduling malleable and nonmalleable paral- lel tasks. In Daniel D. Sleator, editor, Proceedings of the 5th Annual ACM- SIAM Symposium on Discrete Algorithms*, ACM Press, January 1994.