Cluster team

Table des matières

1 Présentation du groupe

1.1 Quentin “Underflow” de Laroussilhe

Je suis très motivé dans la réalisation ce projet, en effet, cela fait plusieurs mois que je m'intéresse aux techniques de parallélisme et je vais enfin me lancer dans une application concrête pour commencer à appréhender toutes les contraintes qu'imposent les architectures parallélisées.

FIGURE 1 – Quentin de Laroussilhe

1.2 Camille Dollé

Bonjour ! Je suis très enthousiaste à propos du projet que nous allons tenter de réaliser lors de ce second semestre. Tout d'abord, c'est un projet à réaliser en C ! Ce projet me permettra donc d'apprendre à manipuler plus en détails les différents aspects du C.

Ce projet comporte également un fort aspect réseau, or je suis une débutante en matière de réseau, contrairement aux autres membres du projet. J'aurai donc l'occasion d'apprendre une montagne de notions sur le réseau lors de ce projet. Le sujet que nous a proposé Underflow m'a tout de suite intéressée, le projet représente pour moi un vrai challenge car il y a une vraie difficulté à le réaliser. D'une part, car il y a beaucoup de connaissances à acquérir avant de les mettre en pratique et d'autre part car le réseau est un domaine (presque) inconnu pour moi.

Le projet représente donc pour moi une compétence de plus à acquérir en informatique bien que notre projet impose beaucoup de recherches et de travail.

1.3 Guillaume “Vermeille” Sanchez

Je suis fasciné par l'intelligence artificielle, et plus généralement les algorithmes de choix. C'est donc tout naturellement que j'ai eu envie de participer à ce projet : répartir intelligemment des tâches entre plusieurs machines. Ce projet présente

FIGURE 2 – Camille Dollé

également des problématiques de parallélisme, qui forcent à penser différemment, à concevoir des systèmes et des algos différents. Le parallélisme est trivial dans un langage fonctionnel pur comme Haskell, langage dont la beauté n'a d'égal qu'une biffle surprise au clair de lune. Comment donc répartir intelligemment un calcul, l'ayant préalablement découpé en composantes fonctionnelles pures ? Voilà la question que je vois au travers de ce projet.

1.4 François Boiteux

Étudiant à EPITA en 2ème année de cycle préparatoire je décide cette année pour le projet libre du second semestre de rejoindre un groupe de projet qui s'interessera à la paralélisation ! C'est un domaine qui offre de multiples possibilités !

D'un côté personnel, j'ai vingt ans car j'ai effectué un an en faculté avant de rejoindre EPITA ! J'aime l'informatique, ma copine, mes amis et c'est ce qui compte le plus pour moi. Mes hobbies sont les jeux videos, la musique et les soirées.

Je m'occuperais de la partie serveur du projet ce qui me permettera d'en apprendre certainement plus sur les sockets et les fonctionnements server/client.

1.5 Yoann "Yoone" Bentz

J'ai commencé à m'intéresser à l'informatique depuis que je suis tout petit, et mon père (dont c'est le travail) a beaucoup contribué à m'y initier.

J'ai commencé sérieusement le développement en développant différents sites web, et c'est en grande partie grâce à cela que je suis à EPITA.

Ce projet m'intéresse beaucoup dans la mesure où je suis nouveau dans le monde du réseau, et que celui-ci m'attire beaucoup. Je m'occuperai d'ailleurs de la réalisation d'un serveur en C pour ce projet.

FIGURE 3 – Guillaume Sanchez

FIGURE 4 – boiteu_f

2 La parallélisation

2.1 Pourquoi paralléliser un calcul ?

Certains algorithmes peuvent demander des temps de calcul importants ou très important. Paralléliser un algorithme permet de distribuer les tâches entre plusieurs noeuds de calcul et ainsi diviser le temps nécessaire à sa réalisation.

Par exemple, un algorithme permettant de retrouver un mot de passe à base de sa signature nécessite de tester une à une toutes les combinaisons possibles jusqu'à trouver le hash recherché. Si la fonction de hashage est sûre, il n'existe pas de méthode de recherche plus optimisée. L'intérêt de diviser le calcul si l'on dispose de plusieurs machines pour effectuer un tel calcul est alors évident.

2.2 Les contraintes de la parrallélisation

Pour faire fonctionner un programme sur plusieurs clusters de calcul travaillant en collaboration on doit avoir pensé le programme pour un tel mode de fonctionnement.

2.2.1 Les effets de bords

Les algorithmes itératifs non designés pour fonctionner sur des architectures parallélisées contiennent des effets de bord, c'est à dire que l'opération d'une instruction executée à un instant t dépend des instructions éxecutées précédemment.

Ainsi, si nous possédons plusieurs noeuds de calcul et que nous décidions d'envoyer à chaque noeud une instruction différente aucun parallélisme ne serait possible car chaque noeud devrait attendre que les noeuds en charge des instructions précédentes aient fini d'executer leurs instructions. Cela revient strictement à une éxecution séquentielle des instructions.

La première difficulté est donc de minimiser les effets de bords pour minimiser les temps d'attente pour executer les instructions.

2.2.2 Les problématiques d'accès mémoire

Si tous les noeuds travaillent sur le même espace mémoire, il peut se produire des problèmes d'accès si deux tâches tentent d'accéder au même emplacement.

2.3 Calcul distribué en réseau

En anglais, le terme “cluster” signifie “grappe”. “Computer cluster” désigne donc un regroupement de plusieurs ordinateurs indépendants.

Chaque ordinateur sera appelé de “noeud” et sera utilisé comme un serveur de calcul indépendant. Mais le cluster final sera considérée comme une seule et même entité.

Les objectifs d'une telle configuration sont multiples : dépasser les limitations d'un seul ordinateur, augmenter la disponibilité... En bref, le “computer cluster” sert à obtenir de plus grandes performances.

En effet, la principale utilisation du “computer cluster” est le calcul parallèle.

2.3.1 Comment regrouper la puissance de plusieurs ordinateurs ?

Le procédé est très simple : on relie les ordinateurs via le réseau, en général un réseau local rapide puis on donne du travail à chaque noeud de façon intelligente de façon à respecter les contraintes entrainées par une architecture parralèle pour faire progresser le calcul.

Nous présenterons dans la section suivante l'architecture que nous avons pensé pour notre cluster.

3 Etat de l'art

Nous citerons ici la description d'un des projet de “Network Computing” appelé BOINC. Il existe toutefois d'autres projets de supercomputing dans le monde.

3.1 BOINC

BOINC est l'acronyme de Berkeley Open Infrastructure for Network Computing (Infrastructure non propriétaire de Berkeley dédiée au calcul en réseau). BOINC est une plate-forme de calcul partagé. C'est donc un programme qui permet de faire participer des volontaires à des projets scientifiques susceptibles de faire avancer la science dans différents domaines tels que la médecine, l'astronomie, la physique ou les mathématiques.

BOINC a une puissance totale moyenne de calcul d'environ 6,5 PFLOPS repartie sur environ 800 000 ordinateurs en septembre 2012 (pour comparaison, le supercalculateur le plus puissant à la même époque est le Sequoia d'IBM, qui atteint une puissance de 16,324 PFLOPS).

3.2 Principe de fonctionnement

La plate-forme BOINC suit un modèle simple : chaque projet s'étend sur un serveur central qui exécute une application Master. Les applications sont réparties sur des milliers de petites tâches qui sont envoyées à des machines réparties sur l'Internet, où ils exécutent des applications de type travailleurs. Il n'y a pas de communication entre les travailleurs. Toutes les communications doivent être du travailleur au Maitre, afin de permettre la traversée de NAT et pare-feu. Les applications BOINC sont donc limitées au modèle Master/worker, avec un serveur central chargé de distribuer le travail aux clients BOINC. BOINC suit un protocole de réseau simple, qui demande aux clients d'initier toutes les communications (problèmes de NAT/pare-feu) et de communiquer avec le serveur à chaque fois qu'un client demande plus de travail. Chaque fois qu'un client est inactif et décide d'exécuter plus de travail, il doit contacter le serveur principal trois fois.

4 Architecture envisagée

4.1 Note préalable

Bien que nous ayons mûrement réfléchi sur l'architecture que nous allions mettre en place nous ne pouvons garrantir que le projet final sera implémenté tel que nous l'envisageons actuellement : de nouvelles idées peuvent émerger et les benchmark vont très probablement nous permettre d'obtenir énormément d'informations nous permettant de faire évoluer notre système.

4.2 Architecture système du cluster

Nous avons décidé de réaliser un système de parallélisation de tâche destiné à fonctionner en réseau, sur plusieurs machines. Celui-ci sera basé sur deux types de machines : le dispatcher (ou le maître), chargé de séquencer et d'organiser les différentes étapes de calcul et les noeuds (ou les esclaves) qui effectuent des tâches.

4.2.1 Les noeuds de calcul

Les noeuds seront chargés d'effectuer les tâches envoyées par le dispatcher, puis de faire remonter le résultat associé à chaque tâche.

Contraintes

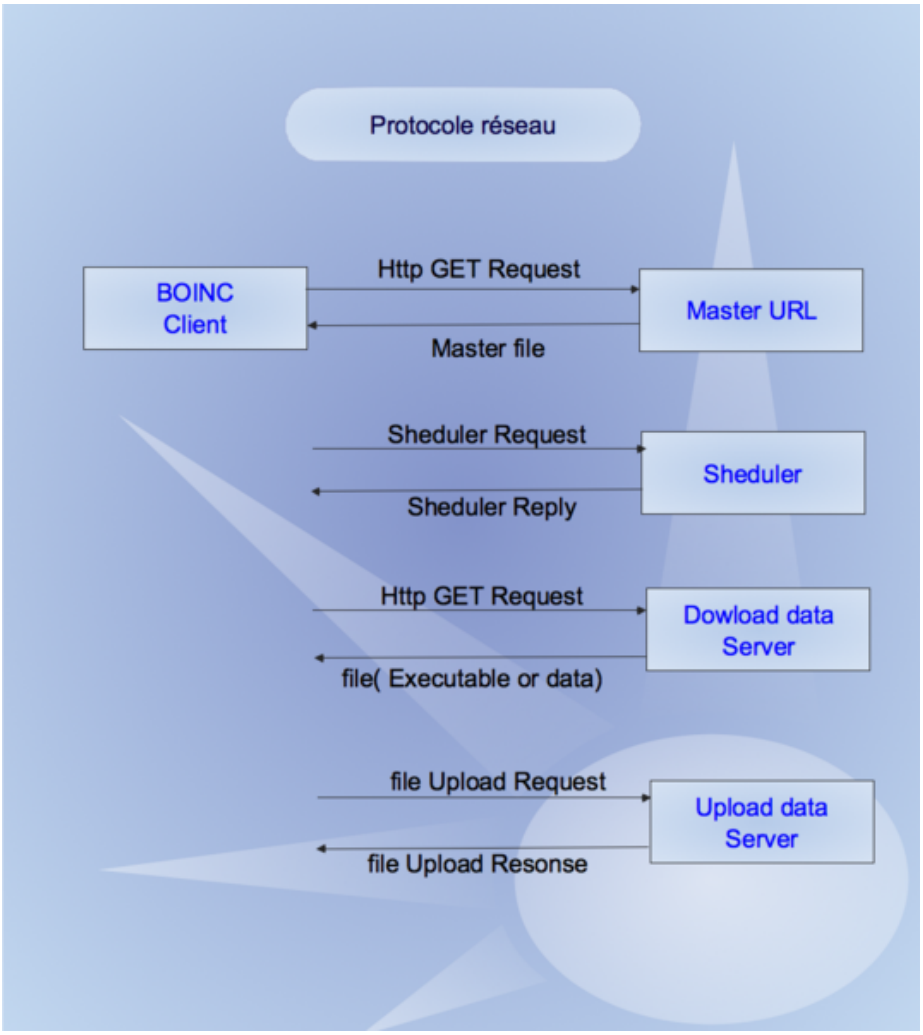

FIGURE 5 – Protocole réseau

FIGURE 6 – Un technicien de My Little Cluster en train de maintenir l'arcihtecture

- Les disques durs étant coûteux et les temps d'accès long, le système d'exploitation doit pouvoir être chargé entièrement en RAM et fonctionner sans disque dur.
- Le dispatcher doit avoir possibilité de contrôler l'ensemble des noeuds facilement (reboot, vérification de l'état, ...)
- Le kernel doit permettre de multi-threader les processus pour tirer profit des processeurs multi-coeurs
- La stabilité du noeud importe peu, en cas d'erreur ou de crash du kernel le protocole sera capable de remettre le noeud en fonctionnement

Pour ces raisons nous nous orienterons probablement sur une distribution linux sur mesure minimaliste comprenant le kernel et le strict minimum (libc, un shell et quelques binaires tout au plus).

4.2.2 Le dispatcher

Le dispatcher a pour rôle d'organiser les tâches nécessaire au calcul. Il va donc répartir le travail entre chaqun des noeuds et récupérer les différents résultats.

L'architecture système du dispatcher est beaucoup moins stricte que celle des noeuds de calcul : il n'est pas en soit un goulot d'étranglement pour la vitesse de calcul et il n'y en a qu'un seul par cluster.

Nous développerons donc un dispatcher fonctionnant sur Linux. Il offrira la

possibilité de sauvegarder les tâches déja calculées pour reprendre le calcul en cas d'interruption.

4.3 Schedulding des tâches

Le dispatcher tiendra à jour une liste des tâches.

Nous penchons pour l'instant sur un round robin avec une simple liste circulaire. Cette liste comprendra un nombre statique de tâches et sera remplie au fur et à mesure. A chaque fois qu'une tâche est distribuée, nous passerons à la suivante. Cela permet, si un noeud ne répond pas, de réattribuer la tâche plus tard.

Lorsque le dispatcher reçoit un résultat, il enlève la tâche correspondante de la liste et il ajoute à la "fin" de la liste circulaire (c'est à dire juste avant l'élement pointé) une nouvelle tâche qui sera distribuée prochainement.

Ce schedulding est minimaliste et servira de "prototype". Nous nous réservons la possibilité de le faire évoluer pour gagner en performance en rajoutant nottament des options de timeout et de priorité d'execution.

4.4 Protocole réseau

Nous utiliserons un protocole qui ne permet pas de communication entre les noeuds pour faciliter le schedulding des tâches.

Le dispatcher contrôlera l'ensemble des nodes et pourra les faire rebooter, suivre leur activité.

Il s'occupera de distribuer les tâches à chaque fois qu'un noeud lui demande.

Lorsqu'un noeud est inactif, il demandera une tâche via un packet réseau, et le dispatcher répondra en envoyant une tâche à effectuer et des paramètres pour l'execution. Une fois le résultat calculé il remontera le résultat au dispatcher et le noeud sera à nouveau considéré comme inactif (voir les spéficitations détaillées du protocole).

5 Restrictions

Ce projet étant à réaliser dans un cadre scolaire, nous sommes tenus de respecter certaines consignes quant au dévelopmment du projet.

Le projet est à réaliser à partir du mois de janvier jusqu'à la fin du mois de mai et l'avancement du projet sera présenté lors de trois soutenances (en mars, avril, et mai). Lors de chaque soutenance, nous serons tenu de fournir : - un rapport de soutenance, qui expliquera ce que nous avons fait pour arriver au résultat

```
Paquet description
==================

This paquet format is used to communicate between the dispatcher and the nodes.

=====  =================
Bytes  Block description
=====  =================
8      client ID (MAC ?)
1      cluster ID
1      opcode
8      size of the block
size   data
=====  =================


Opcodes
=======

These opcodes determines the nature of the instruction.  The range 0-63 is
reserved for the protocol itself and the range 64-255 is used by extensions.

======  ===========   ============================
Opcode  Name          Description
======  ===========   ============================
0x00    NOP           do nothing
0x01    ASK_TASK      ask to be assigned to a task
0x02    PUSH_TASK     push a task to a node
0x03    SEND_RESULT   send the result back
======  ===========   ============================
```

FIGURE 7 – Spécifications du procole de communication (draft)

présenté - un site web, qui permettra entre autres de télécharger les sources du projet.

Le projet devra être développé sous OpenSUSE (Linux) et être codé en C. Nous présentons ici un cahier des charges avec une certaine répartition des tâches, sachant que des modifications pourront être effectuées au fur et à mesure de l'avancée du projet.

6 Répartition des tâches

Voici la répartition des différentes tâches à accomplir pour ce projet. Chaque membre du groupe a ses propres buts à accomplir avant les présentations de chaque soutenance.

Tâches	**Assignation**	**Palier 1**	**Palier 2**
OS des noeuds	François, Camille	Kernel Linux opérationnel	NULL
Outils	Yoann, Quentin	Boot réseau	Interface web de gestion cluster
Protocole réseau	Quentin, Yoann	Bases (envoi et traitement des tâches)	Options avancées (time-outs, monitoring etc...)
Bibliothèque network computing	Tout le monde	NULL	Lib node et dispatcher comprenant l'ensemble des fonctions pour utiliser le cluster
Applications parallèles	Guillaume	Application de démo simple (ex : calcul de hash)	Application de démo évoluée (ex : algorithme génétique)