-

XtremWeb-CH : Une plateforme Global Computing pour les applications de haute performance

http://www.xtremwebch.net

Nabil Abdennadher, Régis Boesch

Laboratoire d'Informatique Industrielle
Ecole d'Ingénieurs de Genève
Université des Sciences Appliquées-Suisse Occidentale

-

XtremWeb-CH a été financé par la réserve stratégique de la HES-SO

-

TABLE DES MATIERES

-

I. Introduction

Depuis la fin de la dernière décennie, le paysage du calcul de haute performance a changé de manière radicale. Basé initialement sur l'utilisation de super-calculateurs parallèles et/ou vectoriels dotés d'environnements de développement spécifiques et propriétaires, les consommateurs de puissance de calcul sont en train d'adopter une approche différente. Celle-ci consiste à profiter du développement du réseau internet pour déployer les applications de calculs intensifs sur les ordinateurs qui lui sont connectés et qui sont équipés de ressources souvent sous exploitées : faible taux d'utilisation des processeurs, espaces disques non utilisés, etc. La nouvelle tendance n'est donc plus d'extraire à l'aide d'applications exécutées par lots (batch), le maximum de puissance d'une machine dédiée, mais plutôt d'extraire, à faible coût et à l'aide d'applications interactives, une puissance de calcul raisonnable à partir d'une plateforme pouvant, potentiellement, fournir de grandes puissances de calcul. Ce concept est connu sous le nom de Global Computing (GC).

La majorité des projets de Global Computing ont adopté une architecture centralisée avec un fonctionnement Maître-Esclaves : SETI@home [1], Entropia [2], United Devices [3], Parabon [4], XtremWeb [5], etc.

Une extension naturelle du GC consiste à distribuer d'avantage le traitement et le "degré décisionnel" de sorte à éviter toute forme de centralisation. Les modèles Clients/Serveur et Maître/Esclaves seraient donc abandonnés au profit d'un modèle plus décentralisé. Ce concept, connu sous le nom de Peer-To-Peer (P2P), a été utilisé de manière particulièrement réussi dans le domaine de partage de fichiers. Les projets les plus connus sont Napster, Gnutella [6] et Freenet [7]. En effet, l'échange de données en général et de fichiers en particulier s'apprête bien à ce modèle. Toutefois, l'utilisation du P2P dans le domaine du calcul intensif laisse apparaître plusieurs problèmes tant théoriques que pratiques. Les algorithmes d'ordonnancement dynamiques d'applications parallèles/distribuées restent difficilement distribuables. L'idée du P2P Computing va aussi à l'encontre des politiques et techniques de sécurité largement utilisées de nos jours sur Internet : Parefeux (Firewalls), adresses NAT (Network Address Translation) et adresses IP dynamiques. Rappelons que l'objectif de ces techniques est de protéger les serveurs de toute utilisation abusive volontaire ou involontaire de la part des clients. Le réseau Internet se trouve alors partitionné en plusieurs zones protégées et donc incapables de coopérer mutuellement. Les problèmes liés au développement d'un vrai environnement P2P dans le contexte du calcul intensif restent encore ouverts.

Ce document décrit un environnement de Global Computing qui tend vers un système P2P et qui s'appelle ***XtremWeb-CH (XWCH)***. Développé à partir du prototype ***XtremWeb (XW)*** de l'Université d'Orsay (France), XWCH tente d'enrichir XW afin qu'il s'approche le maximum possible des concepts du P2P : décentralisation du traitement et des prises de décision ainsi que le développement de modèles symétriques. En effet, dans les systèmes P2P, les nœuds sont sensées jouer le rôle de clients et de serveurs en même temps. Bien que encore utopique, cette idée sera retenue comme ligne directrice dans le projet XtremWeb-CH.

Ce document est organisé comme suit : le paragraphe II présente le prototype XW tel que développé par le Laboratoire de Recherche en Informatique (LRI) de l'Université d'Orsay. Le paragraphe III détaille l'environnement XWCH et les nouveaux concepts qu'ils proposent par rapport à XW. Le paragraphe IV présente les expérimentations effectuées lors de l'évaluation de XWCH. Enfin, le paragraphe V donne les perspectives de cette recherche.

-

II. XtremWeb (XW)

Le projet XtremWeb (XW), développé initialement à l'Université d'Orsay (France) [5] vise à concevoir et développer un environnement de Calcul Global académique et pluri-disciplinaires. L'environnement XtremWeb est une plate-forme généraliste, orientée calcul intensif. La conception d'XtremWeb a été guidée par une volonté de disséminer la technologie logicielle utilisée. XW est sensé permettre à des centres de recherche, des Universités et des Industriels d'installer et d'utiliser leur propre système de calcul Pair à Pair (Peer-to-Peer : P2P) pour leurs travaux de recherche ou pour la production de calcul. L'un des objectifs de la plateforme est d'imposer le moins de contraintes possibles sur l'environnement à mettre en oeuvre autour d'XtremWeb et les applications à exécuter. A la différence de projets comme SETI@HOME, XtremWeb n'est pas dédié à une application particulière, mais configurable pour toute application.

Le système est construit uniquement à partir de standards éprouvés et des logiciels libres de type *open source* comme les langages C++, Java, le langage de scripts PERL, le protocole XML RPC, le gestionnaire de base de données MySQL et le serveur Web Apache. L'utilisateur d'XtremWeb peut ainsi compter sur un accès facile à ces logiciels et une pérennité du système. XW ne peut être comparé à un super-calculateur parallèle, et ce compte tenu de plusieurs facteurs parmi lesquels on peut citer les trois suivants :

- le nombre de ressources considérées (ordinateurs) est beaucoup plus grand dans un système P2P,
- les ressources sont par essence volatiles (peuvent se déconnecter à tout instant),
- les performances de communication sont généralement faibles comparés aux réseaux utilisés dans les machines parallèles.

II.1 Architecture générale

L'architecture générale d'XtremWeb est fortement centralisée (Figure 1) : un master organise les calculs sur des ordinateurs distants, anonymes et volontaires, géographiquement distribuées. Dans ce qui suit, ces ordinateurs seront appelés *workers*. A la demande d'un client qui désire exécuter son application, le master XW affecte celle-ci à un worker en lui transmettant son code et ses données d'entrée. A la réception des résultats, le master XW les renvoie au client. Dans sa version originale, XW supporte des applications mono-module (par opposition aux applications distribuées composées de plusieurs modules communicants).

A la différence de systèmes comme *Gnutella* [6] ou *Freenet* [7] le système n'apparaît pas identique quel que soit le point de vue que l'on prenne : point de vue du serveur ou point de vue des workers.

Une application est constituée de son code binaire (exécutable) et de ses données d'entrée représentées par un fichier compressé. L'application est écrite de sorte à ce qu'elle puisse lire ses données à partir d'un fichier d'entrée et stocker ses résultats dans un fichier de sortie. Le nom de ces fichiers est fixé par l'application elle-même. L'utilisateur peut spécifier la ligne de commande utilisée lors de l'exécution de l'application. Au moment de sa prise en charge par XW, l'application est sélectionnée par l'ordonnanceur (scheduler) du master pour être affectée à un worker. Son code binaire et ses données d'entrées sont alors transmis au worker. A la fin de l'exécution, les résultats sont stockés dans un fichier de sortie, compressés, puis envoyés vers le master XW.

-

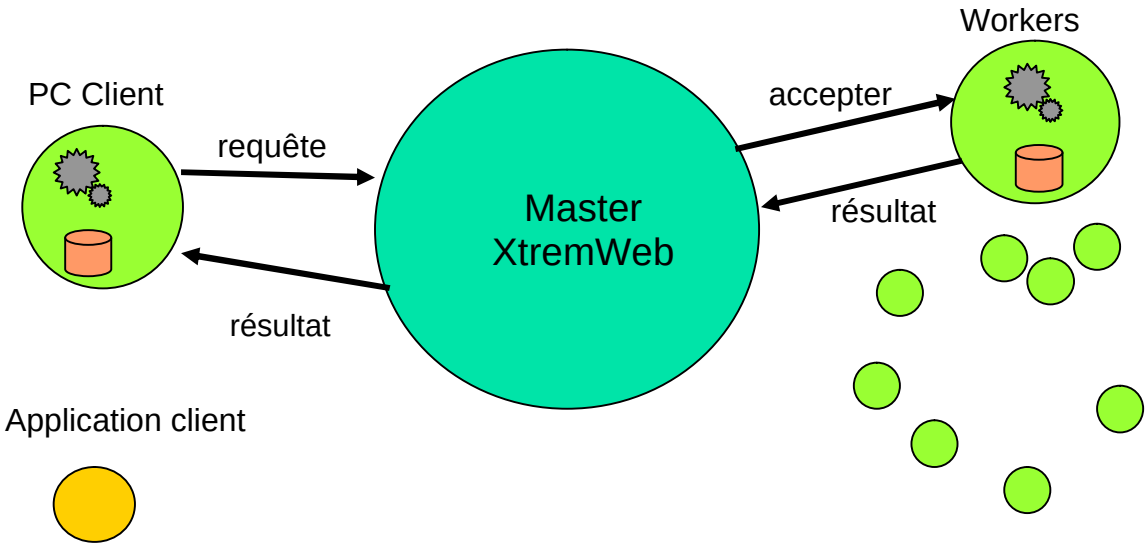


Figure 1 : Architecture de XtremWeb

En supposant qu'XW est installé dans un répertoire appelé *xw*, l'arborescence gérée par le système est la suivante (Figure 2):

- */xw/application/bin* : contient les codes exécutables. Plusieurs codes binaires peuvent correspondre à la même application, ils correspondant, par exemple, à des systèmes d'exploitations différents ou à des processeurs différents.
- */xw/application/user/pid1/* : ce répertoire contient les données d'entrée et de sortie de l'application *application* lorsqu'elle est lancée par l'utilisateur *user*. *pid1* désigne l'identificateur du processus qui correspond à cette application lorsqu'elle est affectée à un worker. Cet identificateur unique est généré automatiquement par le système. Il est donc possible d'avoir plusieurs répertoires pid_i sous le même répertoire */xw/application/user/*. Il est aussi possible d'avoir plusieurs répertoires $user_i$ (au dessous du répertoire */xw/application/*) qui correspondant à des utilisateurs différents.

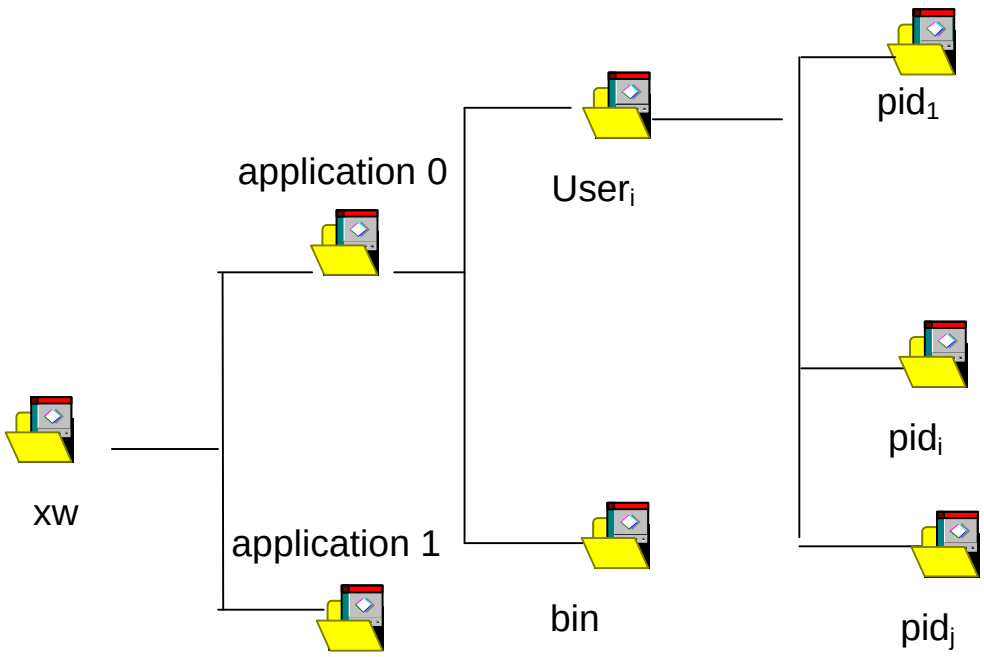


Figure 2: Arborescence des répertoires sous XW

-

II.2 Architecture du worker

Pour qu'un environnement de Calcul Global soit largement accepté auprès du public, il est indispensable que le logiciel *worker* de XW respecte la "propriété privée" du propriétaire de la machine. A cet effet, le worker doit être configurable, non-intrusif et sécurisé.

Le propriétaire d'une machine worker doit pouvoir décider du moment où sa machine lance une exécution et quelles sont les ressources (CPU, taille mémoire, espace disque) qui seront exploitées au cours de cette exécution.

La disponibilité d'un worker dépend de :

- la présence d'un utilisateur (détectée par l'activité clavier/souris)
- l'existence de tâches non- interactives (détectée par l'usage du processeur, de la mémoire ou des E/S)

Le propriétaire définit une politique de disponibilité en indiquant, pour chaque ressource, les seuils qui provoqueraient le lancement ou l'interruption de la participation au calcul global. Un cas particulier est la ressource réseau, qui est directement gérée par la couche de communication : Le worker s'adapte à une déconnection temporaire en sauvant dans une file d'attente les communications à effectuer. Lorsque le worker peut à nouveau contacter le serveur, il reprend les communications en attente. Ce mode est appelé "calcul off-line".

La protection de l'ordinateur d'un utilisateur suggère que l'exécution d'une tâche s'effectue dans un environnement virtuel sécurisé, typiquement la machine virtuelle Java. Toutefois, deux raisons incitent au support de l'exécution d'un code natif :

- Beaucoup d'applications scientifiques utilisent des codes écrits en des langages tels que C ou Fortran,
- Les exigences de performance imposent que le code soit natif à la plateforme d'exécution de l'utilisateur.

L'exécution sur le worker doit donc être confinée dans un environnement où les opérations systèmes sont contrôlées pour interdire/autoriser la lecture ou l'écriture de fichiers ou de sockets, et empêcher l'exécution de processus.

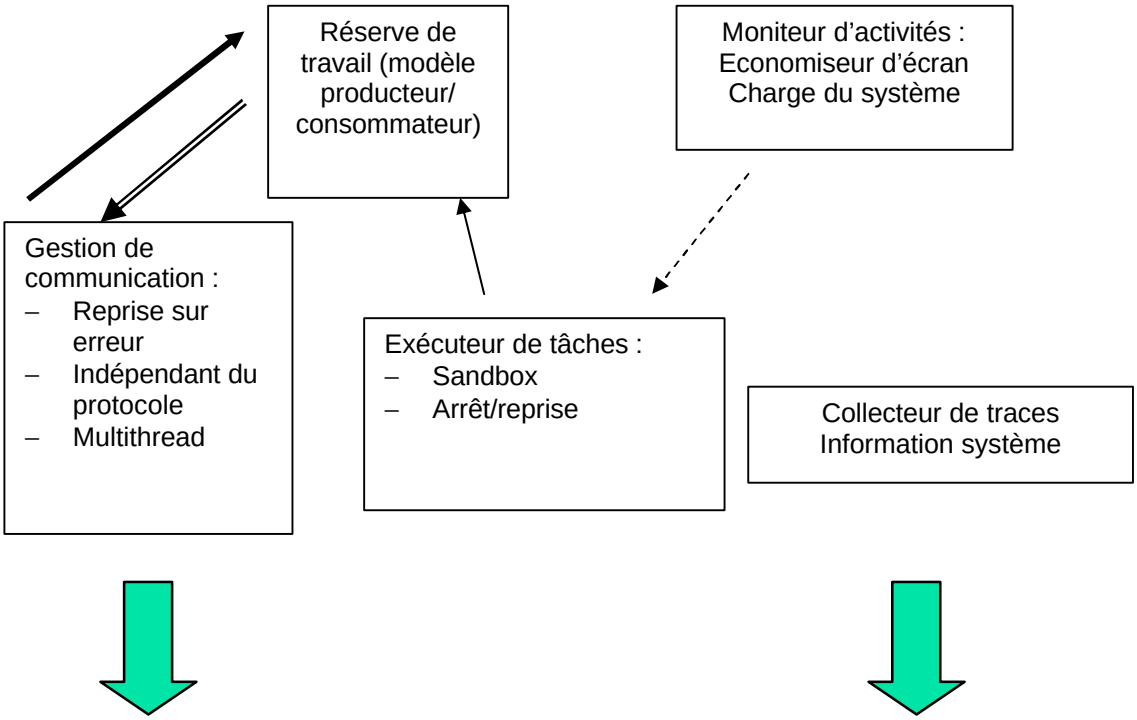
Le worker se connecte à un master suite à une phase d'authentification. Une fois cette phase terminée, toutes les communications sont encryptées y compris le transfert du code au worker. Avant d'exécuter un code, le worker renvoie une signature du fichier binaire et le master vérifie qu'elle est identique à l'original. Enfin les résultats sont aussi encryptés avant d'être rapatriés vers le master de façon à prévenir l'altération des résultats par un tiers. Il est toutefois difficile de garantir que les résultats ne soient modifiés par le worker lui même. Dans ce cas, il appartient aux institutions de fournir, par application, des filtres ou des méthodes statistiques pour éliminer ou atténuer l'influence de résultats manipulés.

II.3 Implémentation du Worker

La figure 3 montre l'architecture du Worker. Celle-ci s'organise autour d'une *réserve de travail* qui est vidée et remplie par un gestionnaire de communication et un exécuteur de tâches. La réserve contient la description des tâches téléchargées par le worker. Le gestionnaire de communication et l'exécuteur de travail sont implémentés en multithead et accèdent à la réserve de travail selon un modèle producteur/consommateur. Ceci permet le parallélisme entre les calculs et les communications (éventuellement plusieurs

communications simultanées), il permet aussi de supporter les machines multiprocesseurs en lançant un exécuteur de tâches par processeur.

D'autre part un thread d'arrière plan, s'exécutant avec une basse priorité, scrute l'activité de l'ordinateur pour appliquer la politique de disponibilité. Lorsque l'ordinateur devient disponible il permet à l'exécuteur de tâches de reprendre une tâche interrompue ou lancer une nouvelle tâche. Lorsque le thread d'observation détecte une augmentation de la charge du système (en excluant celle induite par sa propre activité) ou la présence d'un utilisateur, il ordonne à l'exécuteur de tâches d'arrêter ou de suspendre la tâche en cours.



➡	Fournit les tâches
⟹	Fournit les résultats
⟶	Exécute les tâches en attente
⇢	Utilise/libère la ressource de calcul

Figure 3: Architecture du worker

Le Worker XtremWeb est principalement écrit en Java, les appels à des fonctionnalités spécifiques du système d'exploitation sont écrits en C et intégrés via la Java Native Interface. Le choix du langage Java permet au noyau du Worker XtremWeb d'être facilement porté sur différentes architectures.

II.4 Architecture du Master

Le master XtremWeb est responsable de la prise en charge des applications en. Cette prise en charge consiste à stocker les fichiers binaires des applications compilées et

-

fournir les tâches et les applications aux workers désirant contribuer à l'exécution d'une application particulière.

Pour exécuter une application sous XW il faut fournir un binaire compilé (éventuellement pour plusieurs architectures) et une description de l'application. La description de l'application comprend son nom, la manière dont elle prend en compte ses paramètres et la forme que prennent ses résultats. Lorsqu'une tâche est enregistrée dans la base de données de XW, elle reçoit un dentifiant qui lui est unique.

II.4.1 Répartition des tâches

L'affectation des tâches aux workers est assurée par un module d'ordonnancement. Dans sa version originale, le master XW place les tâches selon une méthode FIFO. Lorsqu'un worker demande une tâche, le master sélectionne d'abord la tâche qui peut être exécutée par ce worker en prenant en considération son environnement d'exécution et l'existence d'un binaire compilé compatible.

Le master détecte aussi les tâches abandonnées en détectant des dépassements de délai (timeout) et les attribue à d'autres workers si nécessaire. Les politiques de sélection et d'affectation de tâches sont configurables dynamiquement. En effet, de nouvelles politiques peuvent être définies en tenant compte des critères spécifiques à l'application et/ou aux ressources.

II.4.2 Implémentation

L'implémentation du master utilise essentiellement des logiciels libres et le langage Java, disponible sur la plupart des systèmes d'exploitation. Le système de gestion de bases de données utilisé est MySQL et les langages utilisés (Java, PHP et Perl) fournissent des accès généralisés aux bases de données (comme Java DataBase Connectivity ou Perl Database Interface).

II.5 Protocole de communication entre Workers et serveurs

Toutes les communications sont à l'initiative du worker. Cette contrainte est indispensable pour contourner les protections réseau (firewall) qui pourraient bloquer les requêtes provenant de serveurs situés à l'extérieur d'un domaine d'administration. Un worker est une machine identifiée par son nom et son propriétaire. Lors de l'inscription, le master vérifie que le nom du propriétaire n'est pas déjà utilisé. Le nom de machine permet au même propriétaire d'inscrire plusieurs machines. L'unicité du nom de machine par rapport à un propriétaire est vérifiée lors de l'enregistrement de la machine.

La figure 4 présente le protocole entre un Worker et un master.

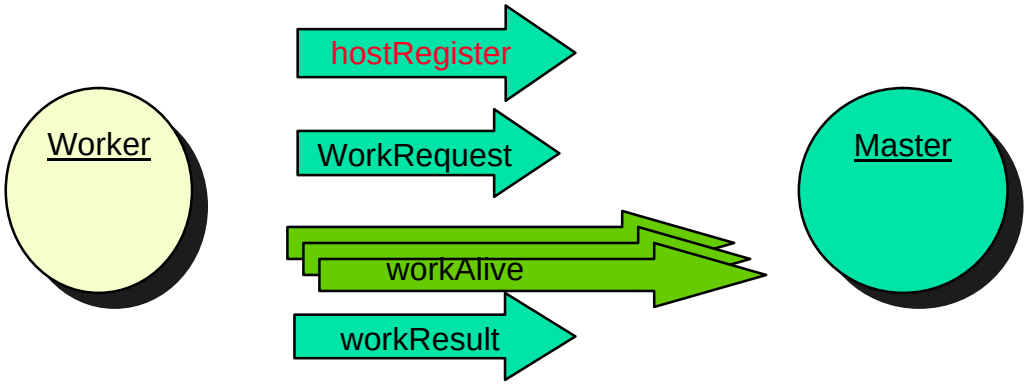


Figure 4 : Protocole de communication entre workers et master

-

Le protocole est constitué de quatre requêtes détaillées ci-dessous:

1. La première requête générée par un worker est *hostRegister*. Elle a lieu avec le dernier master contacté ou le master racine du système (dans le cas de l'existence d'une hiérarchie de masters). Cette première connexion authentifie le master au worker. Le master renvoie un *vecteur de communication* qui spécifie la liste des masters susceptibles de fournir des tâches au worker ainsi que la couche de communication (protocoles et ports) par laquelle ils peuvent être contactés. Dans le cas le plus simple, le master retourne son adresse.
2. Le worker demande alors une tâche au master à travers la requête *workReques*t. Il fournit une description de son environnement d'exécution (système d'exploitation, architecture, etc.) et la liste des applications précédemment téléchargées et répertoriées. En fonction de ces informations, le master sélectionne une tâche et renvoie au worker sa description, ses paramètres d'entrée, son fichier binaire relatif à l'environnement d'exécution du worker.
3. Durant le calcul, le worker invoque périodiquement *workAlive* pour signaler son activité au master. Le master scrute ces appels en permanence pour implémenter un protocole de *timeout*. Si un Worker ne donne pas signe de vie pendant un temps suffisamment long, il est considéré indisponible et sa tâche peut être affectée à un autre worker.
4. A la fin du calcul, le worker renvoie les résultats au serveur à travers l'appel *workResult* de façon à signaler l'aboutissement du travail.

Ce protocole est actuellement implémenté sur les RMI Java et sur SSL.

III. XtremWeb-CH (XWCH)

XtremWeb-CH (XWCH) est une variante de XW qui supporte les applications distribuées composées de plusieurs modules communicants. Lors de la conception et du développement de XWCH, le code et la structure de la base de données de XW n'ont pas été modifiés. A ce titre, XWCH peut être perçue comme une couche au dessus de XW qui prend en compte les communications entre les tâches d'une Application Distribuée (AD). Dans ce contexte, une tâche d'une AD est pour XWCH ce qu'est une application pour XW.

III.1 Structure d'une Application Distribuée (AD)

Une Application Distribuée (AD) est représentée par un graphe flot de données où les nœuds sont les tâches et les arêtes sont les communications inter-tâches (Figure 5). Selon la sémantique de l'AD, les tâches peuvent avoir le même code source ou des codes sources différents. A titre d'illustration, dans la figure 5, les tâches ayant la même forme ont le même code source.

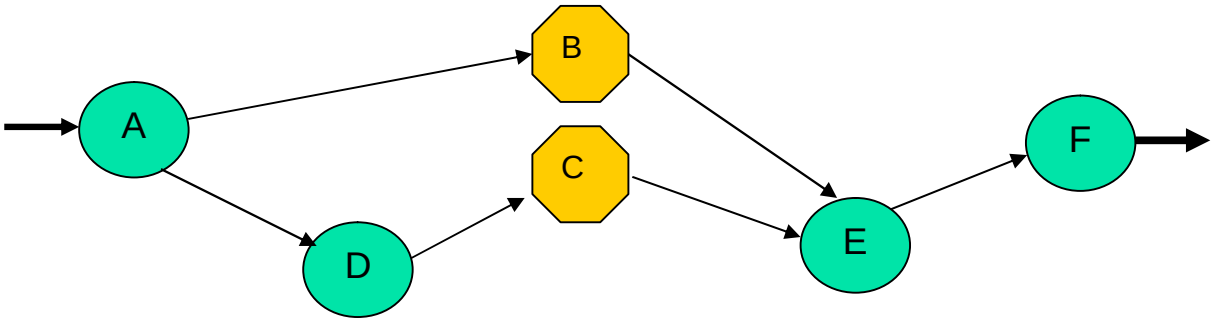


Figure 5: Graphe flot de données modélisant une application distribuée

-

Un module de l'AD est représenté par une application à part entière avec ses binaires et ses processus (voir Figure 2).

La structure de l'AD (codes exécutables, relations de précédence entre tâches, etc.) est décrite dans un fichier XML dont la structure est détaillée dans ce qui suit. La figure 7 donne un exemple de ce fichier qui modélise une application de tri par fusion (*mergesort*). Le graphe flot de données de cette application est représenté par la Figure 6. La figure 8 détaille la syntaxe du fichier XML.

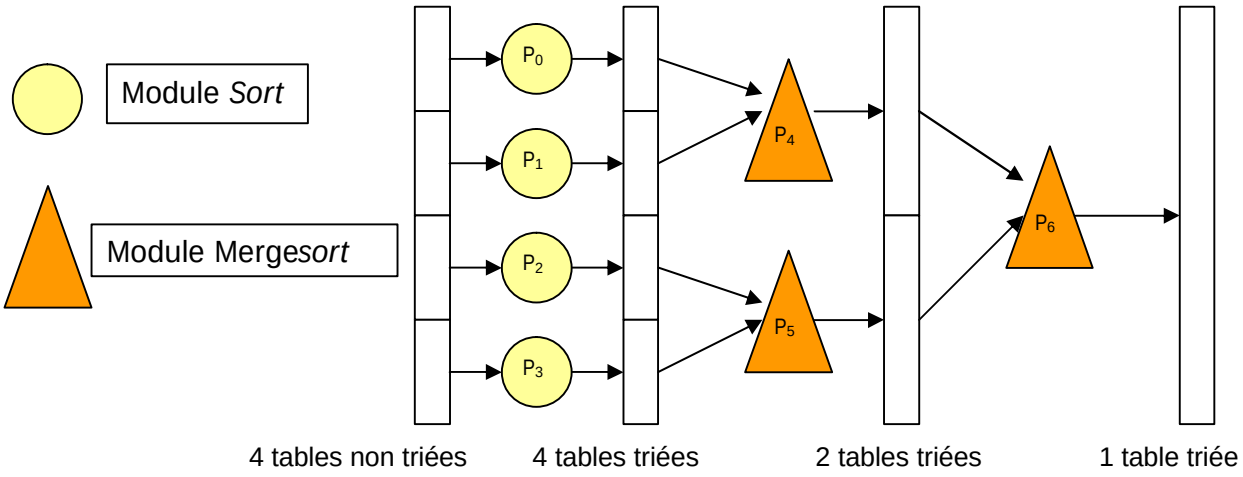


Figure 6: Graphe flot de données modélisant un tri par fusion distribué

Une application peut être composée de plusieurs modules (élément *Module* dans la Figure 8). Un module n'est autre qu'un code source, il peut être représenté par plusieurs versions binaires (élément *Binary* dans la Figure 8). Les relations de précédence entre les tâches sont décrites par les éléments *Task*. Une tâche peut avoir plusieurs données d'entrée (élément *Input* dans la Figure 8) et une seule donnée de sortie (élément *Output* dans la Figure 8). L'élément *cmdLine* indique les arguments utilisés lors de l'exécution. Ce champ est optionnel.

-

```
 <Ordenancement isRelative="1" >
<Application ApplicationDescription="Mergesort by R.Boesch" Client="Boesch_Regis_1">
<Module ModuleDescription="sort" ModuleDirIn="/app/">
<Binary BinaryExecutable="sort" BinarySubFolder="linux" BinaryOsName="Linux"
    BinaryCpuType="ix86" />
    </Module>
<Module ModuleDescription="mergesort" ModuleDirIn="/app/">
<Binary BinaryExecutable="mergesort" BinarySubFolder="linux" BinaryOsName="Linux"
    BinaryCpuType="ix86" />
</Module>
</Application>
<Table>
<Task Description="T0" Application="Mergesort by R.Boesch" ApplicationModule="sort"
    userName="Boesch_Regis_1" DirIn="/0/" FileIn="dir.zip" Final="0">
<Input InputName="in0" />
<Output OutputName="out0" />
</Task>
<Task Description="T1" Application="Mergesort by R.Boesch" ApplicationModule="sort"
    userName="Boesch_Regis_1" DirIn="/1/" FileIn="dir.zip" Final="0">
<Input InputName="in1" />
<Output OutputName="out1" />
</Task>
<Task Description="T2" Application="Mergesort by R.Boesch" ApplicationModule="sort"
    userName="Boesch_Regis_1" DirIn="/2/" FileIn="dir.zip" Final="0">
<Input InputName="in2" />
<Output OutputName="out2" />
</Task>
<Task Description="T3" Application="Mergesort by R.Boesch" ApplicationModule="sort"
    userName="Boesch_Regis_1" DirIn="/3/" FileIn="dir.zip" Final="0">
<Input InputName="in3" />
<Output OutputName="out3" />
</Task>
<Task Description="T4" Application="Mergesort by R.Boesch" ApplicationModule="mergesort"
    userName="Boesch_Regis_1" DirIn="/4/" FileIn="dir.zip" Final="0">
  <Input InputName="out0" />
  <Input InputName="out1" />
  <Output OutputName="out4" />
</Task>
<Task Description="T5" Application="Mergesort by R.Boesch" ApplicationModule="mergesort"
    userName="Boesch_Regis_1" DirIn="/5/" FileIn="dir.zip" Final="0">
  <Input InputName="out2" />
  <Input InputName="out3" />
  <Output OutputName="out5" />
</Task>
<Task Description="T6" Application="Mergesort by R.Boesch" ApplicationModule="mergesort"
    userName="Boesch_Regis_1" DirIn="/6/" FileIn="dir.zip" Final="1">
  <Input InputName="out4" />
  <Input InputName="out5" />
  <Output OutputName="out6" />
</Task>
</Table>
</Ordenancement>
```

Figure 7: Fichier XML modélisant l'application distribuée de tri par fusion (*Mergesort*)

-

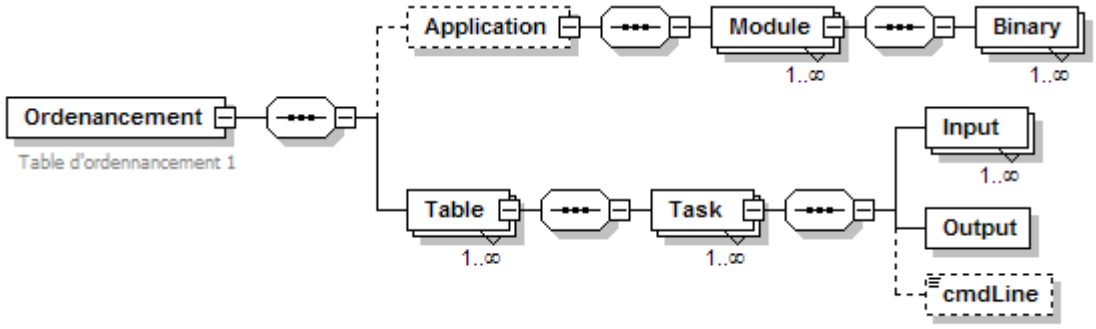


Figure 8 : Structure du fichier XML

De manière plus détaillée, une application est identifiée par son nom (attribut *ApplicationDescription* de l'élément *Application* dans la Figure 7) et un utilisateur (attribut *Client* de l'élément *Application*). L'élément *Application* décrit les modules de l'AD : nom des exécutables, systèmes d'exploitations et types de processeurs supportés, etc.

Chaque module (programme exécutable) de l'AD est décrit par un sous élément *Module* de l'élément *Application*. Cet élément contient les attributs *ModuleDescription* et *ModuleDirIn* ainsi qu'un ou plusieurs éléments *Binary* qui correspondent chacun à un binaire compatible avec un système d'exploitation particulier.

L'attribut *ModuleDescription* indique le nom du module alors que l'attribut *ModuleDirIn* indique le nom du répertoire qui contient le ou les exécutables du module. Chaque exécutable correspond à une plateforme particulière (système d'exploitation, type de processeur, etc.), il est stocké dans un sous répertoire distinct.

Les éléments *Binary* contiennent les attributs suivants :

- *BinaryExecutable* : Nom du fichier exécutable
- *BinarySubFolder* : Nom du sous répertoire (à l'intérieur du répertoire référencé par le champ *ModuleDirIn*) qui contient le fichier exécutable
- *BinaryOsName* : Système d'exploitation concerné
- *BinaryCpuType* : Type du processeur.

L'élément *Table* du fichier XML décrit les tâches du graphe flot de données. Chaque tâche est décrite par un élément *Task* composé des attributs suivants :

- *Description* : indique le nom de la tâche. Celui-ci sera utilisé lors de la visualisation de l'exécution de l'application
- *Application* : Nom de l'application (champ *ApplicationDescription* de l'élément *Application*)
- *ApplicationModule* : Nom du code source qui correspond à la tâche en question. Ce nom doit pointer sur l'un des modules déjà définis auparavant dans les champs *Modules*.
- *userName* : Nom de l'utilisateur (champ *Client* de l'élément *Application*)
- *DirIn* : Répertoire qui contient les données d'entrée de la tâche (fichier compressé)
- *FileIn* : Nom du fichier compressé des données d'entrée (au cas où celles-ci existent)

-

- *Final* : 0 si la tâche a des successeurs, 1 dans le cas contraire. Ce champ est utilisé pour identifier les tâches finales dont les résultats doivent être transmis vers l'utilisateur.

Par ailleurs, chaque élément *Task* comprend un ou plusieurs éléments *Input*, un seul élément *Output* et un élément *cmdLine*. Un 'élément *Input* contient un seul attribut qui référence une donnée d'entrée de la tâche. Cette donnée provient d'une tâche précédente (donnée résultat) ou de l'utilisateur. L'élément *Output* contient un seul attribut *OutputName*. Celui-ci référence le résultat de traitement de la tâche. L'attribut *cmdLine* indique la ligne de commande utilisée lors du lancement de la tâche sur le worker.

Les éléments *Input* et *Output* définissent les précédences entre les tâches.

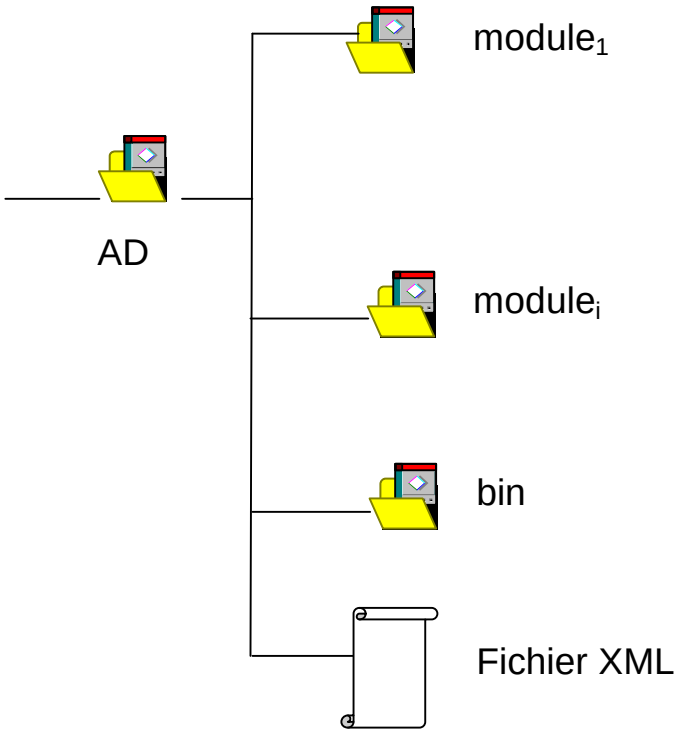


Figure 9: Structure d'une Application Distribuée sous XWCH

Une AD est donc représentée par son fichier XML, ses codes exécutables et ses données d'entrée. Le fichier XML définit les relations de précédence entre les modules et les emplacements des données d'entrée et des codes binaires. Un exemple d'arborescence est donné à la Figure 9. Cette arborescence est soumise à XWCH sous forme d'un fichier compressé.

III.2 Structure de XWCH

Pour pouvoir exécuter une application sous *XWCH*, un utilisateur doit s'inscrire comme client (consommateur) auprès du master *XWCH* qu'il aimerait exploiter. L'utilisateur peut à ce moment déployer ses applications en soumettant tout simplement le fichier compressé qui contient l'arborescence de l'application en question. XWCH se charge alors de décompresser ce fichier et de le transposer dans la structure de XW détaillée au paragraphe II.1 (Figure 2). Les modules de l'application distribuée sont initialement *bloquées* : ils ne peuvent pas être affectés et exécutés par un worker puisque leurs données d'entrées ne sont pas disponibles. Seuls les modules dont les entrées sont données par l'utilisateur peuvent être affectés par l'ordonnanceur à des workers, ils sont à l'état *prêt.* Lorsqu'ils sont en cours d'exécution par un worker, ils sont à l'état *Actif*. Les données d'entrées des modules bloqués sont produites au fur et à mesure par les

modules actifs. Un processus *Espion* (spy) permet de détecter les modules *bloqués* qui peuvent passer à l'état *prêt* et qui peuvent donc être pris en charge par l'ordonnanceur. Deux versions de XWCH ont été développées. La première, appelée XWCH-sMs, gère les communications inter-tâches de manière centralisée, en passant pas le master. La seconde version, appelée XWCH-p2p, permet une communication directe entre les workers sans passer par le master.

III.3 Version centralisée : XWCH-sMs

Dans cette version d'XWCH, les communications entre workers (slaves) transitent par le master qui centralise toutes les communications. Cette version est appelée sMs (slave-Master-slave). Dans l'exemple de la Figure 9, l'AD est composée de deux tâches. La demande d'exécution est envoyée par le client (requête (1)). Celle-ci est prise en compte par XWCH. Un premier worker est sélectionné pour exécuter le module *A* (Accepter (2)). A la fin d'exécution de la tâche, le worker transmet le résultat de son traitement (fichier compressé) au serveur XWCH (Résultat (3)). Ce résultat n'est autre que la donnée d'entrée du module *B*. Le processus espion d'XWCH détecte la disponibilité des données d'entrée du module *B* et le fait passer à l'état prêt. L'ordonnanceur de XW est alors activé pour le l'affecter à un worker. Le master XW lui transmet alors le code binaire et les données d'entrée (Accepter (4)). Les données de sortie du module *A* sont donc transmis deux fois : une fois du worker *A* vers le Master et une fois du Master vers le worker *B*.

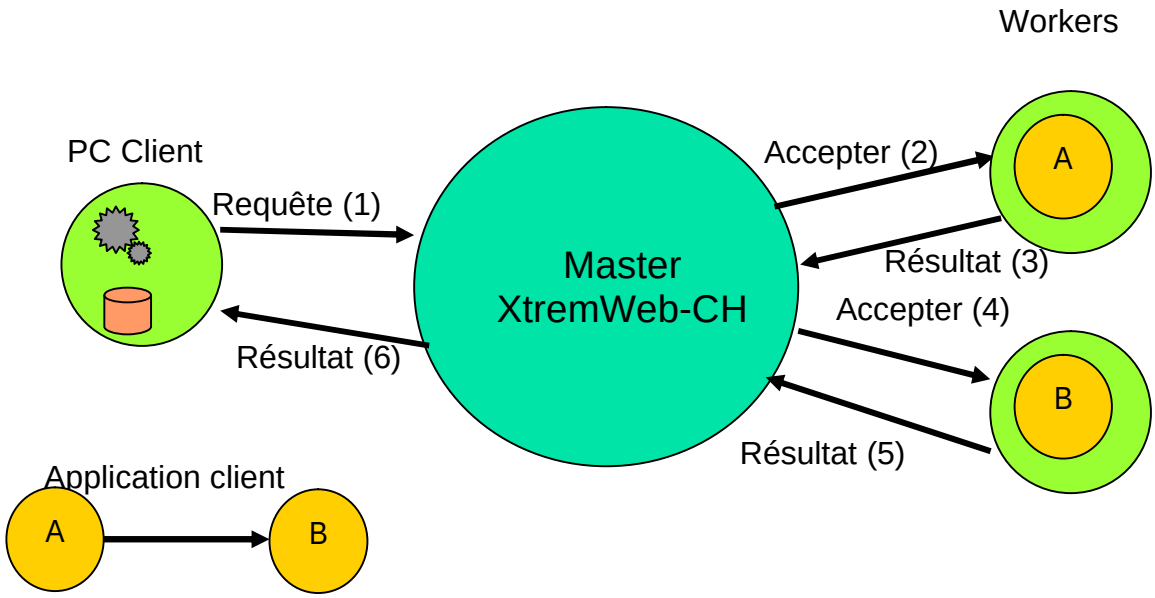


Figure 9 : Structure de XWCH-sMs

Dans la version XWCH-sMs, les workers ne peuvent pas se communiquer directement, ils ne peuvent pas se "voir". Toute communication entre tâches ne peut avoir lieu qu'à travers le master, ce qui engendre inévitablement une surcharge du réseau et pourrait affecter considérablement les performances de l'application distribuée.

III.4 Version décentralisée : XWCH-p2p

Afin de remédier aux lacunes de la version XWCH-sMs, il est nécessaire d'avoir des communications inter-modules directes. En d'autre terme, le worker exécutant le module A dans la Figure 10 (appelé dans ce qui suit worker A) doit pouvoir envoyer ses données résultats vers le worker B, et ce sans passer par le Master. A la fin de l'exécution du

-

module A, le worker envoie tout simplement un signal (signal (3)) au serveur l'informant de :

- la fin d'exécution de A,
- l'emplacement (sur le worker A) et du nom du fichier résultat.

Le master XWCH peut à ce moment débloquer La tâche B et lancer son exécution sur le worker B. Celui-ci reçoit du master le code binaire de B ainsi qu'une information relative à l'emplacement et le nom du fichier d'entrée du module B (adresse IP, chemin et IP). Le transfert de données entre les deux workers peut donc avoir lieu à l'initiative du worker B (Données (4)).

Cette version appelée XWCH-p2p a l'avantage de décharger le Master du routage des données et d'éviter la duplication des communications. Le master ne garde que la gestion de l'ordonnancement des modules. XWCH-p2p s'approche du concept Peer-To-Peer dont l'un des principes et d'éviter tout contrôle centralisé.

La communication directe entre deux workers n'est possible que lorsque le worker hébergeur du résultat (worker A dans la Figure 10) est atteignable par le worker demandeur (worker B). Dans ce cas, la communication directe peut avoir lieu.

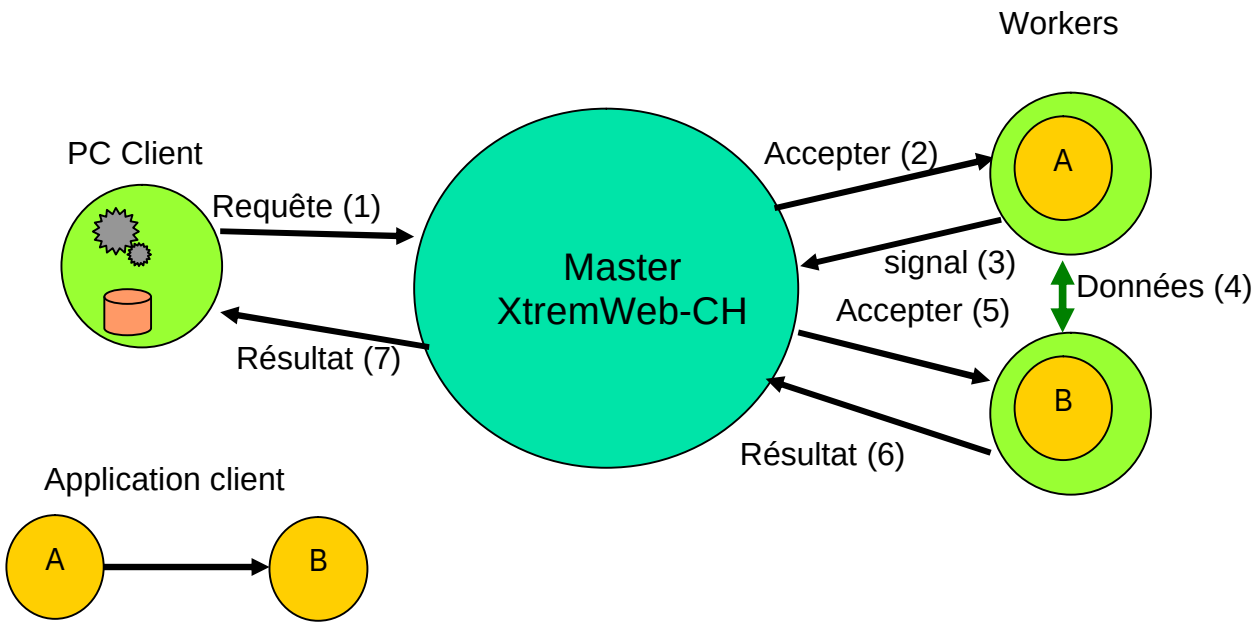


Figure 10 : Structure de XWCH-p2p

Dans le cas contraire (worker A est protégé par un firewall ou fait partie d'un réseau interne protégé par NAT, etc.), une communication directe est impossible. Il est nécessaire de passer par un intermédiaire. Celui-ci peut être le master XWCH lui même. Ce schéma serait donc similaire à la version XWCH-sMs. Toutefois, pour éviter la surcharge du master, la solution retenue consiste à mettre en place une machine relais, appelée «*serveur de données*» qui jouera le rôle de l'intermédiaire et dans laquelle le worker hébergeur stockera ses résultats alors que le worker demandeur ira chercher ses données.

Afin de s'approcher d'avantage du modèle Peer-To-Peer, le serveur de données sera choisi par l'utilisateur lors du lancement de son application. Cette machine doit être accessible par tous les workers contribuant à l'exécution de l'application en question. La possibilité du choix du serveur de données n'est pas encore implémentée dans XWCH.

-

III.5 Mesures et expérimentations

Ce paragraphe présente les mesures expérimentales effectuées dans le cadre du projet XtremWeb-CH. XWCH a été évalué dans le cas d'une application de génération d'arbres phylogénétiques. La phylogénétique est la science qui permet de reconstruire les relations de parenté entre organismes vivants à partir de leurs séquences ADN (Acide Désoxyribo Nucléique). Un arbre phylogénétique (appelé aussi arbre de vie) est alors construit pour montrer les liens de parenté entre les espèces. Cet arbre montre la succession chronologique de l'apparition de nouvelles espèces (et/ou de nouveaux caractères) au cours du temps ainsi que leurs relations de parenté. Dans le domaine médical, la reconstruction d'un arbre phylogénétique relative à une famille de microbes s'avère particulièrement utile pour tracer les mutations qui s'accumulent dans leurs génomes et qui sont dues, entre autres, à la "réaction" du virus aux traitements (anti-biotiques par exemple).

Une multitude d'applications de reconstruction d'arbres phylogénétiques sont utilisées par les chercheurs. Ces applications sont connues pour être très consommatrices de temps processeur, leur complexité est exponentielle (problème NP-complet). Les méthodes approximatives et heuristiques ne résolvent pas définitivement le problème puisque leur complexité reste polynomiale d'ordre 5 ou plus : $O(n^m)$ avec $m > 5$. Toutefois, la parallèlisation de ces méthodes peut s'avérer utile pour réduire les temps de réponse.

La méthode *Tree Puzzle* [8] [9] est l'une des techniques heuristiques utilisées pour la génération d'arbres phylogénétiques. Un algorithme parallèle de cette technique est proposé dans [10] et [11]. Cet algorithme, écrit en C et utilisant les routines de communication MPI a été particulièrement optimisé pour s'exécuter sur un cluster de machines. Dans le cadre du projet XtremWeb-CH, le code de cet algorithme a été repris et porté sur la plateforme XWCH. Les routines MPI ont été remplacées par des transferts de fichiers. Aucune optimisation n'a été faite au niveau du code et de la parallélisation de l'algorithme. Rappelons que le but n'est pas de développer une version optimisée et adaptée à la plateforme XWCH mais de valider les choix retenus en ce qui concerne le développement du prototype XWCH.

La donnée d'entrée de la méthode *Tree Puzzle* est représentée par l'ensemble des séquences ADN des différentes espèces à classer. Une séquence d'ADN est représentée par une chaîne de quelques centaines de caractères. L'algorithme génère une structure représentant l'arbre phylogénétique des espèces données en entrée. La description de la méthode *Tree Puzzle* n'est pas détaillée dans ce document. Il convient toutefois de donner une présentation du graphe de cette application (Figure 11).

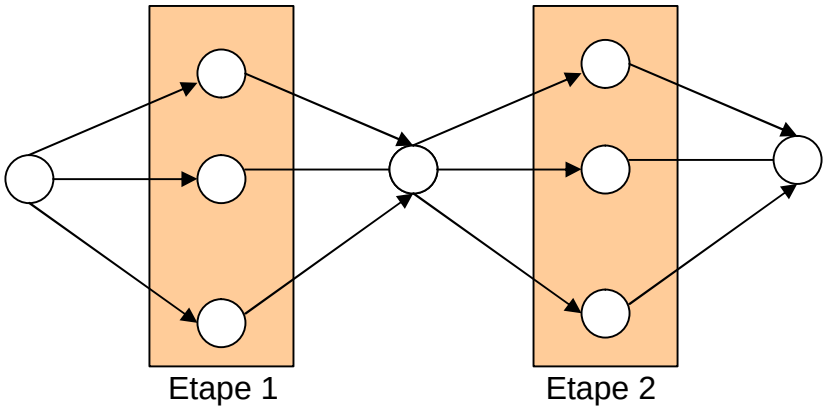


Figure 11 : Graphe de l'application *Tree Puzzle*

-

Les tâches de l'étape 1 (*resp.* Etape 2) ont le même code. Le nombre de ces tâches est fixé lors du développement. Dans la version développée pour ce projet, le nombre de tâches de l'étape 1 est égal à *n* – 3. *n* étant le nombre de séquences d'entrée. Le nombre de tâches de l'étape 2 est variable, mais ne peut jamais dépasser le nombre de séquences.

L'application *Tree Puzzle* a été exécutée sous XWCH (version XWCH-p2p) avec deux jets de données différentes : 64 et 128 séquences d'ADN. XWCH a été installée sur plus que 100 machines hétérogènes installées à l'Ecole d'Ingénieurs de Genève et l'Ecole Polytechnique de Lille (France). Une vingtaine de machines tournent sous Windows. Le reste des machines tournent sous Linux. Les processeurs sont de type Pentium 2, 3 et 4.

L'étape 2 de l'application consomme 70% du temps de traitement de la totalité de l'algorithme. De ce fait, les tests effectués se sont intéressés à faire varier le nombre de tâches de cette étape. Dans la figure 12 (*resp.* 13), l'application *Tree Puzzle* a été exécutée en variant le nombre de tâches de l'étape 2 : 8, 16, 32 et 64 (*resp.* 32, 64 et 128). Lorsque le nombre de séquences est égal à 128 (Figure 13), il n'a pas été possible d'exécuter l'application avec un nombre de tâches de l'étape 2 inférieur à 32 (8 et 16). Pour ces deux cas, les temps d'exécution ont été estimés à 3 jours, pour 16 tâches, et une semaine pour 8 tâches. Les workers de la plateforme XWCH n'ont pas tous été utilisés durant l'exécution. En effet, le nombre de tâches de l'étape 2 n'a jamais dépassé le nombre de workers. En ce qui concerne l'étape 1, l'affectation des tâches aux workers est plus rapide que leur exécution. Par conséquent, l'ordonnanceur XW affecte les tâches de l'étape 1 à des workers ayant déjà exécutés le même code (et donc disposant du code binaire) qu'à de nouveaux workers.

L'objectif de ces mesures n'est pas de comparer les performances de XWCH face à d'autres alternatives de calcul de haute performance mais de montrer simplement la faisabilité de notre approche. Dans ce contexte, aucune optimisation n'a été apportée à l'algorithme parallèle du *Tree Puzzle* utilisé lors de ces expérimentations. Plusieurs améliorations auraient pu être apportées pour adapter l'algorithme parallèle à la plateforme cible XWCH. Toutefois, les ressources nécessaires pour implémenter ces adaptations n'ont pas été mises à disposition dans le cadre de ce projet. Une parallèlisation plus fine de l'algorithme *Tree Puzzle* permettrait de diminuer les temps de réponses et d'utiliser plus de workers. L'apport du parallèlisme serait alors plus visible dans les courbes des Figures 12 et 13.

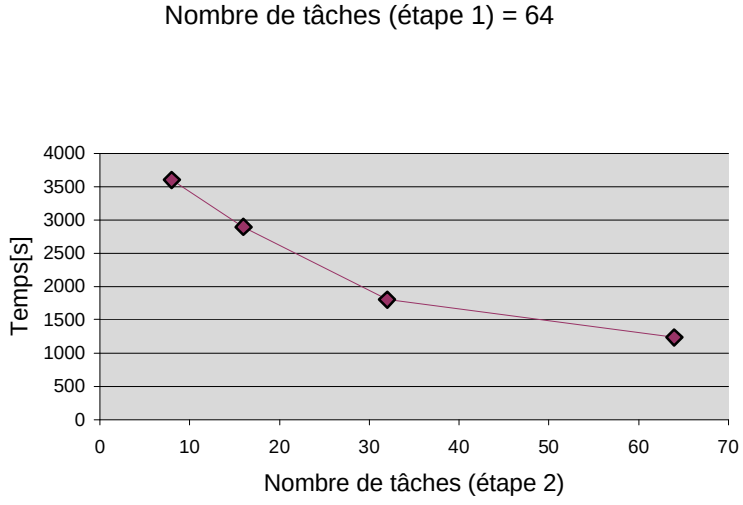


Figure 12 : Temps d'exécution du *Tree Puzzle*.

-

Nombre de séquences = 64

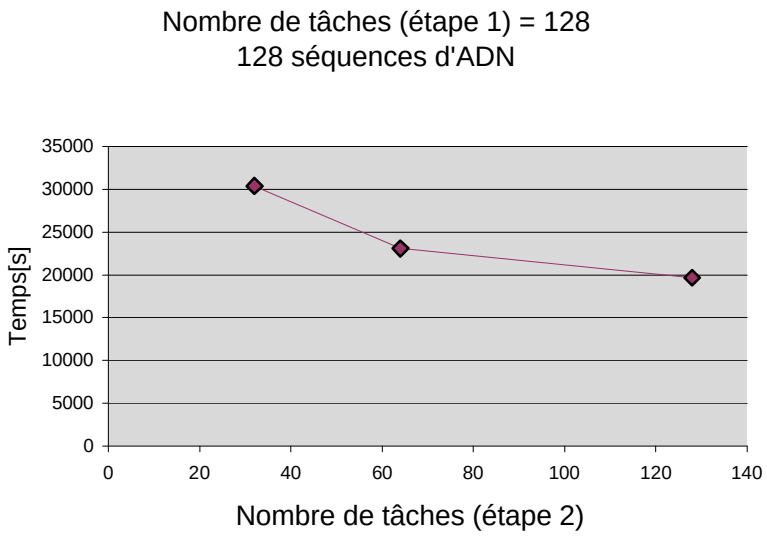


Figure 13 : Temps d'exécution du *Tree Puzzle*.

Nombre de séquences = 128

Par ailleurs, des mesures on été effectuées pour évaluer le trafic généré par XWCH sur le réseau. La figure 14 montre le trafic (exprimé en nombre de bits) en sortie généré par le master XWCH et transitant sur le réseau durant une période de 4 heures et 30 minutes. Durant cette période, deux applications ont été exécutées, elles correspondant aux deux pics observées lors des phases II et V. Ces mesures ont été effectuées grâce à un analyseur de trafic. L'axe des abscisses (x) représente le temps tandis que l'axe des coordonnées (y) représente le nombre de bits émis par le serveur.

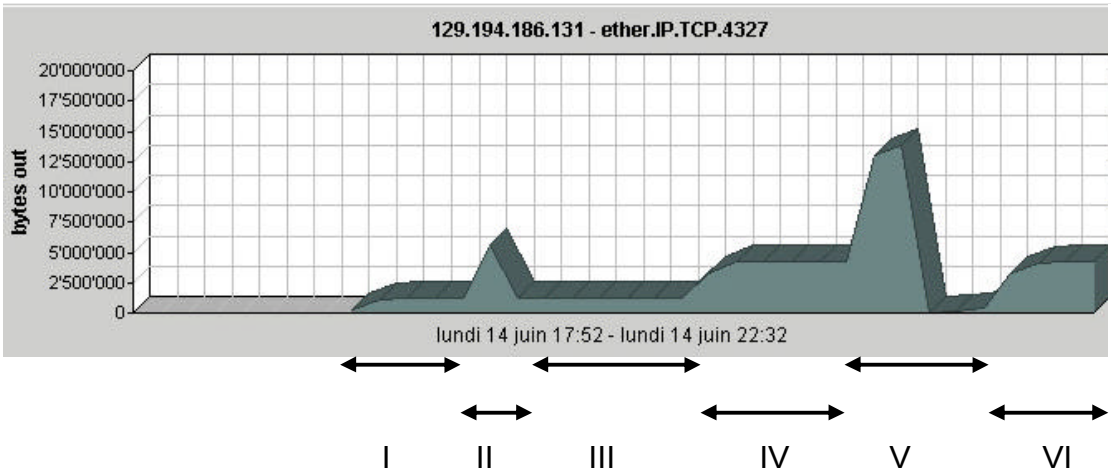


Figure 14 : trafic en sortie généré par le serveur XWCH

La phase I correspond au lancement de 5 workers, le trafic généré par le master correspond aux réponses qu'il génère suite aux appels *workrequest* envoyés par ces workers pour demander du travail au master. Durant cette phase, aucune application n'est déployée. La phase II correspond au lancement d'une application (application de tri par fusion). Ce trafic correspond aux codes et aux données transmis par le master vers les workers. La phase III est similaire à la phase I, elle correspond aux requêtes *workrequest*

-

envoyés par les 5 workers après la fin d'exécution de l'application. La phase IV correspond au lancement simultanée de 24 workers. La phase V correspond au déploiement de la même application de tri par fusion mais avec une taille de données d'entrées plus grande. Le pic en début de cette phase schématise la transmission de codes et de données effectuée par le master. Lorsque les workers exécutent les tâches qui leur ont été affectées, le trafic sortant du master est quasiment nul. En effet, les communications ont lieu directement entre les workers. Ce creux n'apparaît pas dans la première exécution compte tenu de la durée d'exécution courte de l'application. La phase VI est similaire à la phase IV.

Notons enfin que ces mesures ont permis de calculer le trafic moyen (à vide) généré en sortie, par worker, au niveau du master. Ce trafic est de 3.24 kbits/s.

IV. Perspectives et Conclusion

Les tests effectués pour tester XWCH ont permis de valider les choix adoptés dans le cadre de ce projet :

- La version p2p de XWCH a permis de décentraliser la communication et de réduire les temps de d'exécution par rapport à la version sMs. Ce choix va dans le sens du concept Peer-To-Peer dont le principe est d'éliminer tout contrôle et toute centralisation.
- La modélisation d'une application distribuée/parallèle par un fichier XML a permis de la décrire efficacement. Elle permettra, dans l'avenir, de mieux intégrer les évolutions futures. Grâce à cette modélisation, il sera possible de développer un outil de génération automatique du fichier XML, et ce à partir d'une description graphique de l'application. Notons dans ce contexte qu'un premier prototype a été d'ores et déjà développé à l'Ecole d'Ingénieurs de Fribourg [12].

Plusieurs travaux, tant théoriques que pratiques, peuvent constituer la suite de ce projet :

- Un ordonnanceur distribué peut être conçu et mis en place afin de décentraliser d'avantage le master XWCH et s'approcher plus du modèle p2p,
- La génération du fichier XML peut être faite automatiquement en fonction de l'état de la plateforme XWCH : nombre de workers disponibles, état du réseau, etc. Le nombre de tâches serait alors connu juste avant l'exécution et la granularité du parallèlisme est déterminée en fonction du nombre de workers. La structure du fichier XML peut être améliorée de sorte à éviter les duplications inutiles et pesantes dans le cas de grandes applications (centaines et/ou milliers de tâches)
- L'asymétrie du réseau Internet et les politiques de sécurité adoptées par les administrateurs des systèmes connectés empêchent souvent les workers de se communiquer de manière directe. L'une des études pouvant constituer la suite de ce travail consiste à concevoir et implémenter des procédures de détection des workers protégés et de relayer les communications directes inter-workers.

-

V. Bibliographie

[1]: http://setiathome.berkeley.edu/

[2]: http://www.entropia.com/

[3]: http://www.ud.com/home.htm

[4]: http://www.parabon.com/

[5]: www.xtremweb.net

[6]: KAN G., « *Peer-to-Peer: harnessing the power of disruptive technologies* », Chapitre Gnutella, O'Reilly, Mars 2001.

[7]: http://freenet.sourceforge.net/

[8]: http://biowulf.nih.gov/apps/puzzle/tree-puzzle-doc.html

[9]: http://www.tree-puzzle.de/

[10]: http://www.dkfz.de/tbi/tree-puzzle/

[11]: *Heiko A. Schmidt*, Phylogenetic Trees from Large Datasets, 'Ph.D.' in Computer Science, Düsseldorf, Germany, 2003.

[12]: *Tuan-Anh Nguyen, Pierre Kuonen and Omar Abou Khaled*, XtremWeb Portal. A web portal to the Global computing infrastructure. MISL. University of Applied Sciences of Fribourg. January 2004