Solution générique de calcul GRID exploitant des messageries instantanées (Java / Python, XML, XMPP / IRC)

Réalisé par Joffrey Hérard

Responsable : Olivier Flauzac

2016-2017

Table des matières

1 Introduction

Sujet : Solution générique de calcul GRID exploitant des messageries instantanées (Java / Python, XML, XMPP / IRC) Durant ce TER, il a été demandé la mise en place d'un système de calcul reparti entre plusieurs machines avec l'évaluation de possibilités d'exécutions ou non par la machine cible, il fallait aussi évaluer quels échanges allaient être réalisés par les acteurs durant une exécution type et ceci avec le protocole XMPP ou IRC. Il sera détaillé dans ce rapport l'ensemble des échanges ainsi que l'ensemble des gestions d'erreurs déployées.

2 Les Acteurs

Pour commencer, nous avons deux genres d'acteurs pour chaque travail différent disponible :

- Un fournisseur de travail/Provider, est unique pour chaque travail à l'instant t.
- Des travailleurs/Workers, de 1 à n, n défini par le problème. Chaque Provider est possiblement exécuté sur n'importe quel système d'exploitation tout comme chaque worker.
- Un serveur XMPP qui sert de support à la messagerie instantanée avec Openfire.

3 Les Échanges

Voici la liste des différents messages qui transitent à travers une exécution type.

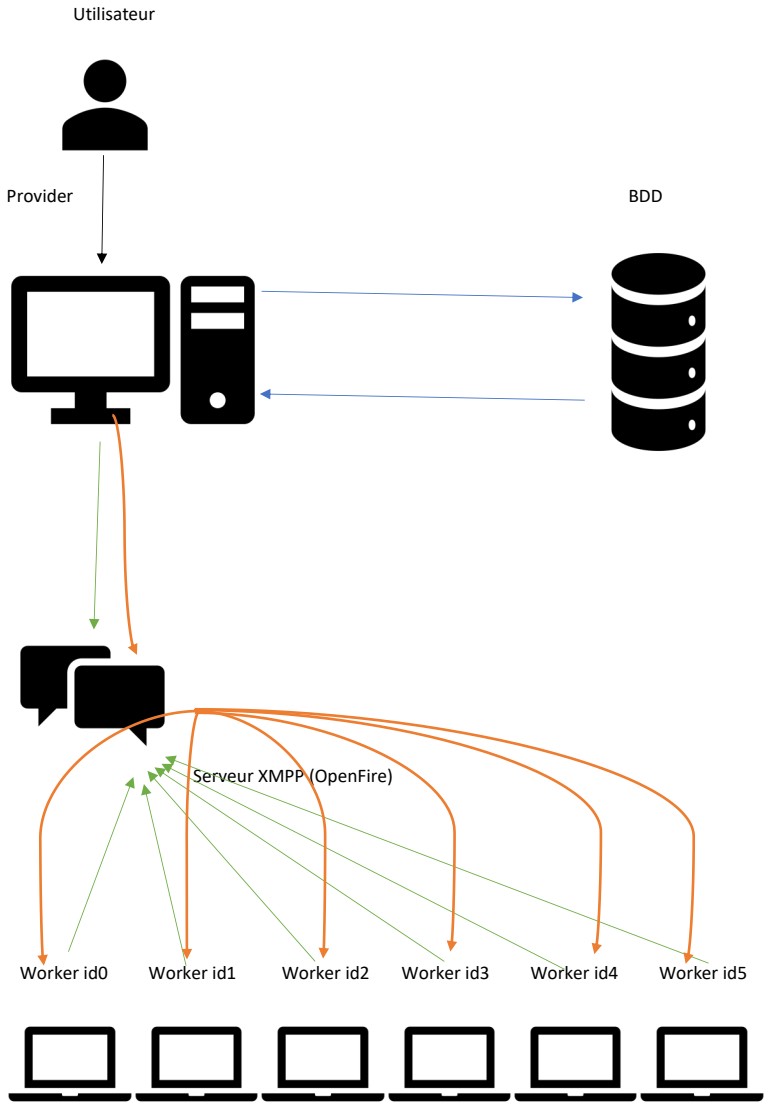
1. Nous avons en premier le message de type "ENVOI JOB", il contient :
   — L'identifiant du problème,
   — Le code des contraintes,
   — Le code à exécuter,
   — La ligne de commande pour l'exécuter.
   — Le nom du fichier à exécuter.
2. Ensuite lorsque le worker est prêt, il le signale par un message qui contient juste dans une chaîne de caractère "Je suis prêt".
3. Il y a enfin le message qui renvoi le résultat "REPONSE JOB" celui-ci contient :
   — L'identifiant pour savoir si le code a pu être exécuté.
   — L'identifiant du problème.
   — La valeur du retour de l'exécution.
   — Le Code de contraintes, si on a pas pu l'exécuter.
   — Le Code exécutable, si on a pas pu l'exécuter.
   — La Ligne de commande associée, si on a pas pu l'exécuter.
   — Le nom du fichier à exécuter.

Voici la liste des fichiers schéma XML associés ainsi que leurs locations au sein du projet :
— "ENVOI_JOB" = ../Schema_XML/ENVOI_RECEPTION.xsd.
— "REPONSE_JOB"= ../Schema_XML/JOB_REP.xsd.

4 Modélisation

4.1 Schéma général

4.1.1 Schéma global d'envoi d'un job



Légendes Envoi Job

Initialisation des Workers = Connection au serveur XMPP

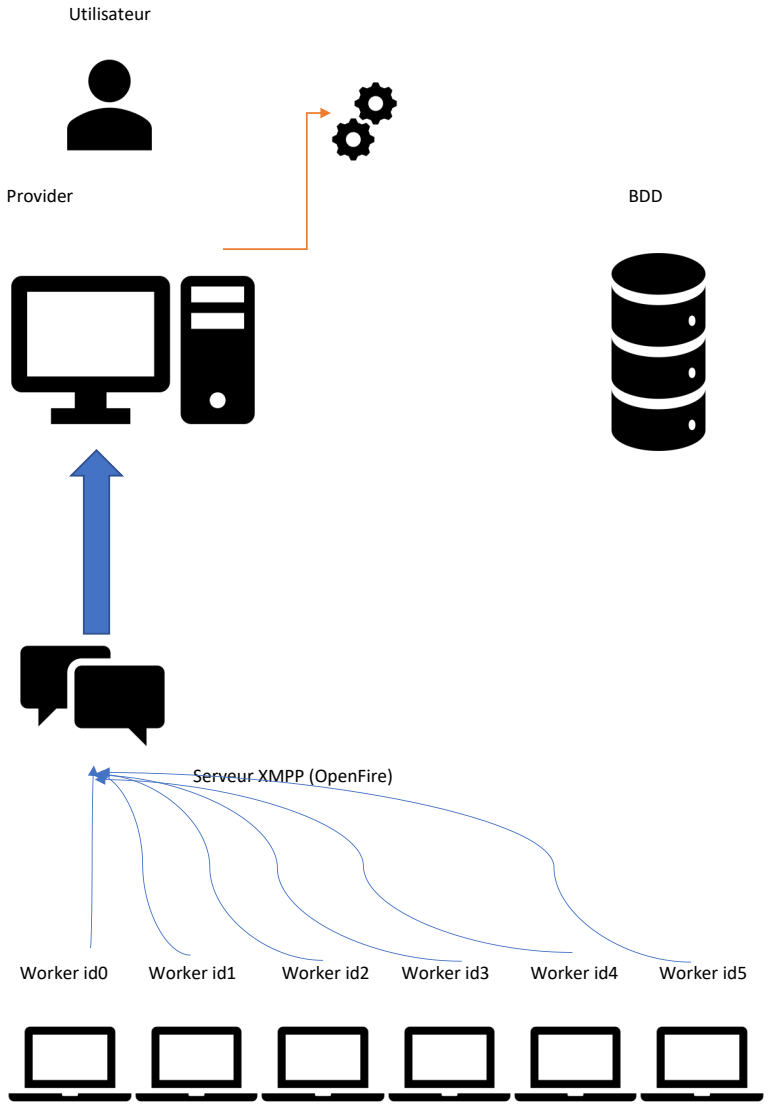
Création d'une chatroom pour un Job de la part du Publisher

Choix du Job par l'utilisateur et on va chercher le schéma XML associé

Distribution du JOB (Split)

Renvoi des resultats

4.1.2 Schéma global d'envoi de réception d'un job



Légende Retour

Flux de retour = calcul

affichage en clair

4.2 Représentation des JOBS

Nous allons dans cette partie du rapport montrer la représentation nécessaire et désirée pour représenter un travail. Donc qu'est-ce qu'un problème ?

- L'identifiant d'un problème, un entier de 0 à n.
- Le code de contraintes, ce code est forcément un code Perl avec un code de retour bien particulier 0 pour non exécutable et 3 pour exécutable et donc que l'on peut exécuter.
- Le type du fichier par exemple ".c , .cpp, .cc, .java , .pl etc.."
- Le code d'exécution : On peut l'exécuter si le code contraintes l'a validé et cela peu importe son langage.
- La ligne de commande pour exécuter le code, par exemple "perl monfichier.pl"

```
<xs:schema>
        <xs:element name="JOB">
                <xs:complexType>
                        <xs:sequence>
                                <xs:element name="nom_fic" type="xs:string"/>
                                <xs:element name="code_Perl" type="xs:string"/>
                                <xs:element name="code_exec" type="xs:string"/>
                                <xs:element name="cmd" type="xs:string"/>
                                <xs:element name="rang" type="xs:Integer"/>
                                <xs:element name="build" type="xs:string"/>
                         </xs:sequence>
                 </xs:complexType>
        </xs:element>
</xs:schema>
```

4.3 Représentation des fichiers de lignes de commandes

Chaque ligne de commande doit être représentée comme elle se doit de respecter ces 2 règles simples :

1. Après la commande de type perl, python. ./, il doit être suivi d'un @,
2. La commande puis une ','

Si ces règles ne sont pas respectées, le "parsing", se faisant sur ce fichier par une expression régulière, ne s'appliquera pas correctement

```
perl@ calcul.pl 2 3,
perl@ calcul.pl 3 3,
perl@ calcul.pl 4 3,
perl@ calcul.pl 5 3,
perl@ calcul.pl 26 3,
perl@ calcul.pl 29 3
```

Exemple de Langford_mono12.dc :

```
.@/langford 12
```

Exemple de nQueen14.dc :

```
python @nQueen14.py
```

La raison principale de la présence de l'arobase c'est qu'il va falloir reconstruire la ligne de commande pour ajouter le chemin correspondant. La raison de la présence de la virgule c'est la ligne de commande personnalisée d'un worker séparée d'une autre.

4.4 Les différentes fonctions principales

4.4.1 Contraintes

L'exécution d'un script de contrainte se fait avec les objets Runtime et Process, on obtient le résultat du script avec la fonction waitFor()

```
#!/usr/bin/perl
use v5.14;

my $osname = $^O;


if( $osname eq 'MSWin32' ){
  print"We are on windows";
  # work around for historical reasons
  exit(1);
} else {
    print "Test si on est sur Mac\n";
    if ($osname eq 'darwin') {
        print "We are on Mac OS X ...\n";
        exit(2);
    }
    else {
        print "Test si on est sur Linux\n";
        if ($osname eq 'linux') {
            print "We are on Linux...\n";
            exit(3);
        }
    }
}
```

4.4.2 Séparation

La fonction split peut se résumer en plusieurs étapes : Dans un premier temps, on récupère tous les nicknames et JIDs de chaque utilisateur de la chatroom, pour chacun d'entre eux on leur envoi un fichier XML personnalisé avec chacun une tâche bien distincte en respectant le schéma associé. Pour avoir une trace on sauvegarde chaque fichier xml envoyé dans le dossier JOB_SEND

```
1  public int split(ArrayList<identity> Liste_user,int Nombre_Participants,
2    String ProblemeCourant,XmppManager xmppManager,String choix)
3  {
4    for(int i=0;i<Nombre_Participants-1;i++)
5    {
6      String buddyJID = Liste_user.get(i).getId();
7      String buddyName = Liste_user.get(i).getName();
8
9      try {
10       xmppManager.createEntry(buddyJID, buddyName);
11
12       /* Recherche de la liste des exec*/
13
14       DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
15       DocumentBuilder builder = factory.newDocumentBuilder();
16
17       File fileXML = new File("DB_JOBS/"+choix);
18
19       Document xml = builder.parse(fileXML);
20
21       Element root = xml.getDocumentElement();
22
23       XPathFactory xpf = XPathFactory.newInstance();
24
25       XPath path = xpf.newXPath();
26
27
28
29       String expression = "/JOB/code_exec";
30
31       String strexec = (String)path.evaluate(expression, root);
32       System.out.print("DEBUT STR EXEC");
33       System.out.print(strexec);
```

```
34        System.out.print("FIN STR EXEC");
35
36        expression = "/JOB/code_Perl";
37
38        String strperl = (String)path.evaluate(expression, root);
39
40        expression = "/JOB/cmd";
41
42        String strcmd = (String)path.evaluate(expression, root);
43
44
45        String delims = "[,]";
46        String[] tokens =strcmd.split(delims);
47        System.out.print("affichage des tokens");
48        tokens[tokens.length-1]=tokens[tokens.length-1].substring(
49        0, tokens[tokens.length-1].length()-1
50        );
51        for(int j=0;j<tokens.length;j++)
52          System.out.println(j+" : "+tokens[j]);
53
54        final Document document= builder.newDocument();
55
56        final Element racine = document.createElement("JOB");
57        document.appendChild(racine);
58        final Element exec = document.createElement("exec");
59        exec.appendChild(document.createTextNode(strexec));
60
61        final Element contraintes = document.createElement("contraintes");
62        contraintes.appendChild(document.createTextNode(strperl));
63
64
65        final Element cmd = document.createElement("cmd");
66        cmd.appendChild(document.createTextNode(tokens[i]));
67        final Element id = document.createElement("id");
68        id.appendChild(document.createTextNode(""+i));
69        racine.appendChild(id);
70        racine.appendChild(contraintes);
71        racine.appendChild(exec);
72        racine.appendChild(cmd);
73
74        final TransformerFactory transformerFactory = TransformerFactory.newInstance();
```

```
75        final Transformer transformer = transformerFactory.newTransformer();
76        final DOMSource source = new DOMSource(document);
77        final StreamResult sortie = new StreamResult(new File("JOB_SEND/XML_send_"+i));
78        transformer.transform(source, sortie);
79
80        String Probleme_individuel=FileToString("JOB_SEND/XML_send_"+i);
81
82        xmppManager.sendMessage(Probleme_individuel, buddyName+"@"+xmppManager.NOM_HOTE);
83
84        }catch (Exception e) {
85        // TODO Auto-generated catch block
86        e.printStackTrace();
87        }
88        // Ici on fait apelle a la fonction split
89
90    }
91
92    return 0;
93  }
```

4.4.3 Exécution

L'exécution d'un script d'exécution se fait avec les objets Runtime et Process, on attend le résultat du script avec la fonction waitFor() et bien entendu on a un résultat chiffré dans un fichier.

```
1  #!/usr/bin/perl
2  use v5.14;
3
4  exit $ARGV[0] +$ARGV[1] ;
```

4.4.4 Construction du résultat

La fonction build est simplement une addition de chaque résultat reçu petit à petit

```
String from = message.getFrom();
String body = message.getBody();
System.out.println(String.format("Received message '%1$s' from %2$s", body, from));
// on regarde l'en tete du message apparente a un message xml reponse tel que lid est 1 si r
//si imposible a executer dans le cas echeant un troisieme champ correspond a lid du job non
//et 0 si envoie travail

// on procede donc au build un script ici un peu fictif mais pour prolonger pour voir si ca
// ici on va juste sommer les results choses assez simple

String regex="[,]";
String[] en_tete = body.split(regex);

// indice 0 = en tete indice 1 = res
if(Integer.parseInt(en_tete[0])==1){
retour_Providing+=Integer.parseInt(en_tete[1]);
recu++;
if(recu==envoyer)
{
  travail_terminer=true;
}
```

4.5 Description d'une exécution quelconque

4.5.1 Exécution cote Provider

Voici un déroulement classique coté Provider :

1. Un Provider choisi un problème à lancer.
2. Une fois le problème lancé une Chatroom comportant le nom du problème+Providingroom est créée sur le domaine.
3. Le Provider attend un nombre suffisant de Workers en fonction du problème (champ rang du XML).
4. Une fois ce nombre atteint, on récupère chaque identifiant et on envoi à chaque worker son job ainsi que la ligne de commande.
5. On attend d'avoir reçu un message de type "JOB_Reponse" autant de fois que de Workers capablent de l'exécuter.
6. On affiche enfin le résultat.

4.5.2 Exécution cote Worker

Voici un déroulement classique coté Worker

1. On s'identifie.
2. On choisi une salle de travail .
3. Une fois connecté on applique la même routine : A savoir, on exécute chaque script de contrainte, si ils sont validés, on exécute le fichier exécutable avec la ligne de commande associée.

4.5.3 Exécution d'un ajout

1. On demande le nom du problème.
2. On demande en entrée le chemin absolu pour un fichier de contrainte.
3. On demande en entrée le chemin absolu pour un fichier exécutable.
4. On demande en entrée le chemin absolu pour un fichier correspondant aux lignes de commande.
5. On demande en entrée un rang qui est égal au nombre de workers nécessaires.
6. Tout ceci est ajouté à un fichier XML nommé nom_du_probleme.xml

4.5.4 Exécution d'une suppression

1. On demande le nom du problème.
2. On supprime le fichier XMl associé.

5 Formatage des fichiers sources décrivant un JOB

5.1 Script de contraintes

5.1.1 Principe

Tous scripts de contrainte doivent répondre à leurs propres contraintes :

1. Être écrit en Perl.
2. Avoir un retour de commande avec "exit(3) ;" pour un retour validant la poursuite du processus.

5.1.2 Exemple

Exemple pour un programme interprété Pour un langage interprété comme le python en voici un exemple :

```
#!/usr/bin/perl

use v5.14;


my $value= system("python -V");

if( $value eq 0)
{
        exit(3);
}
else
{
        exit(0);
}
```

Exemple pour un programme compilé Quand il s'avère que vous devez valider la compilation puis sa future exécution d'un code exécutable tel qu'un code de langage compilé comme le C en voici un exemple pour connaître la démarche à suivre.

```
#!/usr/bin/perl
use v5.14;
use Cwd 'abs_path';
use Data::Dumper qw(Dumper);

my $osname = $^O;
```

```
if ($osname eq 'linux') {
        print "We are on Linux...\n";
        my $str = abs_path($0);
        print $str."\n";
        my $taille = length $str;
        my $taille_fichier = length "contraintes.pl";
        print $taille_fichier."\n";
        print $taille."\n";

         my $path  = substr $str,0,$taille-$taille_fichier;
        print "My path is ; ".$path."\n";
          system("gcc -std=c11 -pedantic -Wall -Wextra -O3  ".$path."langford.c -o ".$path."l
        exit(3);
}
else
{
        exit(0);
}
```

5.2 Script de Commandes

5.2.1 Principe

Cette partie fait aussi une référence, voire une redite avec le section 5-3. Tous scripts de commande doivent répondre à leurs propres contraintes :

1. Après la commande faisant appel à un interpréteur mettre le caractère "@".
2. Dans le cas d'un split à plus de un Worker, utilisez les virgules pour séparer chaque commande appropriée pour chaque Worker.
3. Les caractères évidemment interdits sont :« @, ',', [,] ». En raison d'exploitations d'expressions régulières.

5.2.2 Exemples

Voici quelques exemples :

```
python @nQueen14.py

perl@ calcul.pl 2 3,
perl@ calcul.pl 3 3,
perl@ calcul.pl 4 3,
perl@ calcul.pl 5 3,
```

```
perl@ calcul.pl 26 3,
perl@ calcul.pl 29 3
```

5.3 Script d'exécution

5.3.1 Principe

Tous scripts des exécutions doivent répondre à leurs propres contraintes :

1. Avoir connaissance que toutes les données renvoyées par les Worker seront passées dans le fichier résultat.
2. Écrire le résultat dans un fichier "resultat.txt".

5.4 Script de build

5.4.1 Principe

Tous scripts de construction doivent répondre à leurs propres contraintes :

1. Être écrit en Perl.
2. Avoir connaissance que toutes les données renvoyées par les Workers seront passées en argument.
3. Écrire le résultat dans un fichier "resultatF.txt".

5.4.2 Exemple

```
#!/usr/bin/perl
#---------------------#
#  PROGRAM:  argv.pl  #
#---------------------#

$numArgs = $#ARGV + 1;
print "thanks, you gave me $numArgs command-line arguments:\n";
my $sum;
foreach $argnum (0 .. $#ARGV) {
  $sum=$sum+ $ARGV[$argnum];
}
print $sum;

# Création du fichier '.txt'
open (FICHIER, ">resultatF.txt") || die ("Vous ne pouvez pas créer le fichier \"resultatF.txt
# On écrit dans le fichier...
print FICHIER  $sum ;
```

```
exit 0;
```

6 Contrainte du serveur OpenFire

Il est nécessaire, afin que le programme fonctionne, que le serveur soit accessible par tout le monde de là où il se trouve, il est possible de demander une machine virtuelle avec un serveur OpenFire (ou autre dailleurs) tant qu'il est possible pour un provider d'y être inscrit par un administrateur au grade de modérateur, respectivement pareil pour un worker. Il est important de savoir que là il y a plusieurs limite de OpenFire.

7 Les Erreurs

7.1 Les Problèmes d'exécution

Les problèmes qui peuvent opérer à travers le système sont :

1. Un mauvais nom de domaine.
2. Un problème de Chatroom déjà existante.
3. Un problème d'exécution : aucun worker ne peut exécuter le code, comment le détecter ?

7.2 Les Problèmes réseau

Nous avons plusieurs problèmes liés au réseau quelque soit le protocole utilisé :

1. Latence/Impossible à établir une connexion à la Chatroom.
2. Latence/Impossible à envoyer un message d'un Provider vers un Worker.
3. Latence/Impossible à envoyer un message d'un Worker vers un Provider.
4. Un Worker est déconnecté en plein milieu de sa tâche.
5. Un Provider est déconnecté durant l'attente d'une réponse sur un JOB.

7.3 Gestions des erreurs

7.3.1 Gestions des erreurs sur l'exécution

1. Mauvais nom de domaine → Redemander le nom de domaine jusqu'à validation.
2. Problème de ChatRoom déjà existante → Message d'erreur un problème exactement identique est en cours d'exécution.
3. Problème d'exécution : aucun worker ne peut exécuter le code, comment le détecter ? → Mise en place d'un tableau de variable booléenne au départ initialisé à faux, si un worker renvoi avec une impossibilité d'exécution du code dicté par le code contrainte, alors on met à vrai et on redistribue. Si aucun n'est capable on arrête l'exécution.

7.3.2 Gestions des erreurs sur le réseau

1. Latence/Impossible à établir une connexion à la ChatRoom→ Message qui explicite le fait d'aller voir un Administrateur Réseau.
2. Latence/Impossible à envoyer un message d'un Provider vers un Worker→ Message qui explicite le fait d'aller voir un Administrateur Réseau.
3. Latence/Impossible à envoyer un message d'un Worker vers un Provider→ Message qui explicite le fait d'aller voir un Administrateur Réseau.
4. Un Worker est déconnecté en plein milieu de sa tâche→ Détecteur de présence permis par le protocole XMPP sur une ChatRoom MultiUser.
5. Un Provider est déconnecté durant l'attente d'une réponse sur un JOB→ Évaluation de présence d'un Provider, si aucun alors arrêter le Job en cours, ou mise en place d'un TimeOut.

8 Conclusion

Cette conclusion sera séparée en plusieurs parties : la première concernant l'aspect des problèmes qui resteraient à gérer. La deuxième partie sera concentrée sur la partie XMPP, la troisième sur les jobs à entrevoir, et la quatrième sur une virtualisation des services. Pour finir cette conclusion, un retour sur le TER.

Les problèmes qu'il reste à gérer

1. Comment résoudre une perte réseau de longue durée sur un worker, quelle serait la pertinence d'un envoi de résultat après un certain temps ?
2. Avoir des informations de disponibilité de la bande passante du serveur hôte de messagerie instantanée si l'on souhaite avoir des communications en temps réel.
3. Si le Provider de JOB tombe en panne pendant les calculs que faire, que ce soit un problème réseau ou électrique ?

OpenFire

1. Il reste à évaluer les besoins et la montée en charge associés au serveur XMPP. Il aurait été plus intéressant lors de mes tests, d'utiliser un serveur XMPP externe pour lui faire éprouver des stress tests. Ainsi donc en voir des éventuelles limites qui apporteraient de nouveaux problèmes à gérer.
2. Il y a des machines virtuelles avec des serveurs XMPP déjà configuré c'est une voie qu'il reste à exploiter.

JOB à entrevoir Comme nous en avons déjà parlé durant quelques entrevues, il reste un point sur lequel je n'ai pu me pencher : une utilisation par exemple de l'outil POV-ray ou des tabulations de Golomb. Respectivement, POV-ray s'utilisant en lignes de commandes, seul le support de travail reste à définir, les tabulations de Golomb sont particulières, elles restent du domaine du calculatoire sur les espacements de blocs.

Virtualisation des services Une possibilité qui techniquement était envisageable sur ce sujet de TER, il est possible de lancer des conteneurs par exemple avec Docker. Évidemment lintérêt serait d'avoir des machines personnalisées avec les outils qui serait intéressant. Tout cela j'ai pu les envisager avec l'aide de Guillaume Bourgeois, qui m'a conseillé sur la manière de les gérer, je n'ai évidemment pas développé en profondeur cela dans le désir de ne pas empiéter sur son sujet.

En résumé Le sujet avait beaucoup d'aspects à couvrir, bien entendu tout l'aspect réseau, pour la transmission des messages, l'aspect programmation pour s'adapter aux API XMPP disponibles, et bien entendu l'aspect détudes sur les échanges, et la montée en charges des besoins pour le serveur XMPP.

9 Annexes

9.1 Organisation du dossier du projet

- /
  - /bin
    - Tout les fichiers .class
    - Images
    - Schema
  - /DB_JOBS
    - Calculatoire.xml
    - LangfordMono12.xml
    - NQueenMono8build.xml
    - Tout les probleme.xml
  - /Echantillon_Script_Cmd
    - Toto.dc
    - nQueen14.dc
    - Langford_mono12.dc
    - Tout les probleme.dc
  - /Echantillon_Script_Exec
    - calcul.pl
    - Langford(Dossier comportant les fichiers exemple sur langford)
    - nQueen(Dossier comportant les fichiers exemple sur nqueen)
  - /Echantillon_Script_Perl
    - OSname.pl
    - nqueen.pl
    - langford.pl
  - /JDOM
  - /JOB_REC
    - /DATA_EXTRACT
      - fichier_extraits
    - xml_receive.xml
  - /JOB_SEND
    - XML_send_0.xml
  - /openfire
  - /Rapport
    - TER_Joffrey_Herard.pdf
  - /Schema_XML
    - BDD_JOB.xsd
    - ENVOI_RECEPTION.xsd
    - JOB_REP.xsd

- /smack_3_1_0
- /src
  - Tout les fichiers .java
- README.md

9.1.1 Outils et langages

1. Le projet a été programmé en Java version : "1.8.0_111" sous Eclipse.
2. L'API smack a été utilisé pour mettre en place les échanges sous XMPP.
3. Openfire a été utilisé pour installer un serveur XMPP en local afin d'exécuter tous les test.

9.2 Code

9.2.1 XML

BDD_JOB.xsd

```xml
<xs:schema>
  <xs:element name="JOB">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="nom_fic" type="xs:string"/>
        <xs:element name="code_Perl" type="xs:string"/>
        <xs:element name="code_exec" type="xs:string"/>
        <xs:element name="cmd" type="xs:string"/>
        <xs:element name="rang" type="xs:Integer"/>
        <xs:element name="build" type="xs:string"/>
       </xs:sequence>
     </xs:complexType>
  </xs:element>
</xs:schema>
```

ENVOI_RECEPTION.xsd

```xml
<xs:schema>
  <xs:element name="JOB">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="nom_fic" type="xs:string"/>
        <xs:element name="id" type="xs:Integer"/>
        <xs:element name="contraintes" type="xs:string"/>
        <xs:element name="exec" type="xs:string"/>
        <xs:element name="cmd" type="xs:string"/>
       </xs:sequence>
     </xs:complexType>
  </xs:element>
</xs:schema>
```

JOB_REP.xsd

```
<xs:element name="id_retour" type="xs:Integer"/>
<xs:element name="reponse" type="xs:string"/>
<xs:element name="id" type="xs:Integer"/>
<xs:schema>
  <xs:element name="JOB">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="nom_fic" type="xs:string"/>
        <xs:element name="contraintes" type="xs:string"/>
        <xs:element name="exec" type="xs:string"/>
        <xs:element name="cmd" type="xs:string"/>
       </xs:sequence>
     </xs:complexType>
  </xs:element>
</xs:schema>
```

9.2.2 Fichier de lignes de commande .dc

Les exemple de fichier de commande écrit en texte clair Toto.dc

```
perl@ calcul.pl 2 3,
perl@ calcul.pl 3 3,
perl@ calcul.pl 4 3,
perl@ calcul.pl 5 3,
perl@ calcul.pl 26 3,
perl@ calcul.pl 29 3
```

9.2.3 Perl

Les exemple de fichier de contrainte écrit en Perl langford.pl

```
#!/usr/bin/perl
use v5.14;
use Cwd 'abs_path';
use Data::Dumper qw(Dumper);

my $osname = $^O;

if ($osname eq 'linux') {
  print "We are on Linux...\n";
  my $str = abs_path($0);
  print $str."\n";
  my $taille = length $str;
  my $taille_fichier = length "contraintes.pl";
  print $taille_fichier."\n";
  print $taille."\n";

   my $path  = substr $str,0,$taille-$taille_fichier;
  print "My path is ; ".$path."\n";
    system("gcc -std=c11 -pedantic -Wall -Wextra -O3  ".$path."langford.c -o ".$path."langfo
  exit(3);
}
else
{
  exit(0);
}
```

nqueen.pl

```
#!/usr/bin/perl

use v5.14;


my $value= system("python -V");

if( $value eq 0)
{
  exit(3);
}
else
{
  exit(0);
}
```

OSname.pl

```
#!/usr/bin/perl
use v5.14;

my $osname = $^O;


if( $osname eq 'MSWin32' ){
  print"We are on windows";
  # work around for historical reasons
  exit(1);
} else {
    print "Test si on est sur Mac\n";
    if ($osname eq 'darwin') {
        print "We are on Mac OS X ...\n";
        exit(2);
    }
    else {
        print "Test si on est sur Linux\n";
        if ($osname eq 'linux') {
            print "We are on Linux...\n";
```

```
21              exit(3);
22          }
23      }
24  }
```