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

Joffrey Hérard

Responsable : Olivier Flauzac

2016-2017

Table des matières

Chapitre 1

Introduction

Sujet : Solution générique de calcul GRID exploitant des messageries instantanées (Java / Python, XML, XMPP / IRC) Durant ce TER, la mise en place d'un système de calcul repartie entre plusieurs machine avec lévaluation de possibilité dexécutions ou non par la machine cible, il fallait aussi évaluer quels échanges allais être réalise par les acteurs durant une exécution type et ceci en avec le protocoles XMPP ou IRC .

Chapitre 2

Les Acteurs

Nous avons donc deux genres d'acteur pour chaque travail différent disponibles
— Fournisseur de travail/Provider, unique pour chaque travail.
— Des travailleurs/Workers, de 1 a n, n définit par le problèmes. Chaque Provider est possiblement exécuter sur n'importe quel système d'exploitation tout comme chaque worker

Chapitre 3

Les Échanges

Voici la liste des différents message qui transitent a 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 a exécuter,
   — La ligne de commande pour l'exécuter.
   — Le nom du fichier à exécuter.
2. Ensuite il y a le message ou le workers signale qu'il est prêt il contiens juste un message pour signale dans une chaîne de caractère " Je suis prêt".
3. Il y a enfin le message qui renvoi le résultat "REPONSE JOB" il contient :
   — L'identifiant pour savoir si le code a pu être exécute.
   — L'identifiant du problème.
   — La valeur du retour de l'exécution.
   — Code de contraintes, si on a pas pu exécuter .
   — Code exécutable, si on a pas pu exécuter .
   — Ligne de commande associe, si on a pas pu exécuter .
   — Le nom du fichier à exécuter.

Voici la liste des fichiers schéma XML associe ainsi que leurs locations au sein du projet :

— "ENVOI_JOB" = ../Schema_XML/ENVOI_RECEPTION.xsd.
— "REPONSE_JOB"= ../Schema_XML/JOB_REP.xsd.

Tout les codes du projet sont présenter en annexe.

Chapitre 4

Les Erreurs

4.1 Les Problèmes dexécution

Les problèmes qui peuvent opérer a travers le système,sont :

1. Mauvais nom de domaine
2. Problème de Chatroom déjà existante
3. Problème dexécution : aucun worker peut exécuter le code, comment le détecter ?

4.2 Les Problèmes réseaux

Nous avons plusieurs problèmes lie au réseaux quelque soit le protocole utilise :

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

4.3 Gestions des erreurs

4.3.1 Gestions des erreurs sur l'exécution

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

4.3.2 Gestions des erreurs sur le réseaux

1. Latence/Impossible a établir une connexion a la Chatroom→ Message qui explicite le fait daller voir un Administrateur Reseaux
2. Latence/Impossible a envoyer un message d'un Provider vers un Worker→ Message qui explicite le fait daller voir un Administrateur Réseaux
3. Latence/Impossible a envoyer un message d'un Worker vers un Provider→ Message qui explicite le fait daller voir un Administrateur Réseaux
4. Un Worker est déconnecte en plein milieu de sa tache→ Détecteur de présence permis par le protocole XMPP sur une ChatRoom MultiUser
5. Un Provider est déconnecte durant l'attente d'une réponse sur un JOB→ Évaluation de présence d'un Provider, si aucun alors arrêter le Job en cours, ou mis en place d'un Timeout.

Chapitre 5

Modélisation

5.1 Connexions

5.1.1 Connexions du/des Provider(s)

5.1.2 Connexions du/des Worker(s)

5.2 Représentation des JOBS

Nous allons dans cette partie du rapport montrer la représentation nécessaire et désirer pour représenter un travail.Donc un problèmes c'est quoi ?

- Identifiant d'un problème, un entier de 0 a n.
- Code de contraintes, ce code est forcement 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.
- Type du fichier par exemple ".c , .cpp, .cc, .java , .pl etc.."
- Code dexécution, peut importe son langage. on peut l'exécuter si le code contraintes l'a valider
- 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>
```

5.3 Représentation des fichiers de lignes de commande

Chaque ligne de commande doit être représenter comme elle se doit c'est a dire la commande puis une ',' sinon, 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
```

5.4 Les différentes fonctions principales

5.4.1 Contraintes

L'execution d'un script de contrainte ce fait avec les objets Runtime et Process on obtient le résultat du script avec la fonctin 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);
        }
    }
}
```

5.4.2 Split

La fonction split peut se résumer en plusieurs étapes, premièrement on récupère tout les nickname et JID de chaque utilisateur de la chatroom ,pour chacun d'entre eux on leur envoi un fichier xml personnalise avec chacun une tache bien distincte en respectant le schéma associe.Pour avoir une trace on sauvegarde chaque fichier xml envoyer dans le dossier JOB_SEND

```
public int split(ArrayList<identity> Liste_user,int Nombre_Participants,
  String ProblemeCourant,XmppManager xmppManager,String choix)
{
  for(int i=0;i<Nombre_Participants-1;i++)
  {
    String buddyJID = Liste_user.get(i).getId();
    String buddyName = Liste_user.get(i).getName();

    try {
      xmppManager.createEntry(buddyJID, buddyName);

      /* Recherche de la liste des exec*/

      DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
      DocumentBuilder builder = factory.newDocumentBuilder();

      File fileXML = new File("DB_JOBS/"+choix);

      Document xml = builder.parse(fileXML);

      Element root = xml.getDocumentElement();

      XPathFactory xpf = XPathFactory.newInstance();

      XPath path = xpf.newXPath();



      String expression = "/JOB/code_exec";

      String strexec = (String)path.evaluate(expression, root);
      System.out.print("DEBUT STR EXEC");
      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  }
```

5.4.3 Exec

L'exécution d'un script dexécution ce fait avec les objets Runtime et Process on obtient le résultat du script avec la fonctin waitFor() bien entendu on parle de calcul chiffrer ,cela peut être aussi un résultat chiffre dans un fichier.

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

5.4.4 Build

La fonction build est simplement une addition de chaque résultat reçu petit a 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;
}
```

5.5 Description d'une exécution quelconque

5.5.1 Exécution cote Provider

Voici un déroulement classique cote Provider :

1. Un Provider choisi un problème a lancer.
2. Une fois le Problème lancer une Chatroom comportant le nom du problème+Providingroom est créer sur le domaine
3. Le Provider attend un nombre suffisant de Worker en fonction du probleme(champ rang du XML)
4. Une fois un nombre de Worker atteints, on récupère chaque identifiant et on leur envoi leur job respectif et ligne de commande respective
5. On attend d'avoir reçu un message de type "JOB_Reponse" autant de fois et distinct que de Workers capable de lexécuter
6. On affiche le résultat

5.5.2 Exécution cote Worker

Voici un déroulement classique cote Worker

1. On s'identifie
2. On choisi un salle de travail
3. une fois connecter on applique la même routine
4. A savoir, on exécute chaque script de contrainte si ils sont valide on exécute le fichier exécutable avec la ligne de commande associe.

5.5.3 Exécution d'un ajout

1. On demande le nom du problème
2. On demande en entrer le chemin absolu pour un fichier de contrainte
3. On demande en entrer le chemin absolu pour un fichier exécutable
4. On demande en entrer le chemin absolu pour un fichier correspondant aux ligne de commande
5. On demande en entrer un rang qui est égale aux nombre de workers nécessaire
6. Tout ceci est ajouter a un fichier XML nomme nom_du_probleme.xml

5.5.4 Exécution d'une suppression

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

Chapitre 6

Formatage des fichiers sources décrivant un JOB

6.1 Script de contraintes

6.1.1 Principe

Tout script de contraintes doit répondre a ces propres contraintes :

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

6.1.2 Exemple

Exemple pour un programme interpréter

Pour un langage interpréter 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 savère que vous devez valide la compilation puis sa future exécution d'un code exécutable tel q'un code de langage compilé comme le C en voici un exemple sur comment faire

```
#!/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);
}
```

6.2 Script de Commandes

6.2.1 Principe

Tout script de commande doit répondre a ces propres contraintes :

1. Après la commande faisant appelle a un interpréteur mettre le caractère "@"
2. Dans le cas d'un split a plus de 1 Worker, utiliser les virguler pour séparer chaque commande approprie pour chaque worker
3. Les caractères évidemment interdit sont : @, ',', [,] .en raison d'exploitation d'expression régulière.

6.2.2 Exemple

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
```

6.3 Script de build

6.3.1 Principe

Tout script de contraintes doit répondre a ces propres contraintes :

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

6.3.2 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;
```

Chapitre 7

Conclusion

Chapitre 8

Annexes

8.1 Organisation du Projet

- /
  - /bin
    - Tout les fichiers .class
    - Images
    - Schema
  - /DB_JOBS
    - Calculatoire.xml
    - Calculatoire2.xml
    - Langford.xml
    - etc..
  - /Echantillon_Script_Cmd
    - Robin.dc
    - Toto.dc
    - etc..
  - /Echantillon_Script_Exec
    - calcul.pl
  - /Echantillon_Script_Perl
    - DitributionContraintes.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

8.1.1 Outils et langages

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