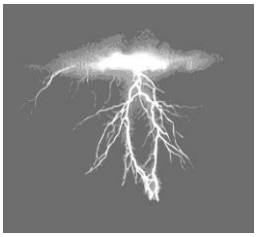

Configurer Apache STORM

avec plusieurs nœuds esclaves

Version 1.1

Rédacteurs V1 : Philippe Lacomme (placomme@isima.fr), Raksmey Phan (phan@isima.fr)

Date : 24 août 2015

Rédacteurs V2 : Philippe Lacomme (placomme@isima.fr), Raksmey Phan (phan@isima.fr), étudiants en projets ISIMA (Soriano Baptiste et Zouggari Yannis)

Date : 23 juin 2016

Installation réalisée sur : Ubuntu 15.04

Environnement : Vmware

Documents à consulter :

https://storm.apache.org/releases/0.10.0/index.html

Licence :

Ce document est une compilation d'information parfois en Anglais ou en Français librement accessibles sur Internet.

Permission vous est donnée de copier, distribuer et/ou modifier ce document selon les termes de la Licence GNU Free Documentation License, version 1.3 ou ultérieure publiée par la Free Software Foundation ; sans section inaltérable, sans texte de première page de couverture et sans texte de dernière page de couverture. Une copie de cette licence en anglais est consultable sur le site suivant : http://www.gnu.org/licenses/fdl.html

Objectifs :

2 objectifs dans ce tutoriel :

- apprendre à configurer plusieurs machines virtuelles

- donner les définitions de bases des différentes architectures

Introduction

Un cluster se compose (figure 1) :

- d'un nœud Maître qui héberge Nimbus, ui et zookeeper ;

- d'un ensemble de nœud esclaves qui hébergent un Supervisor.

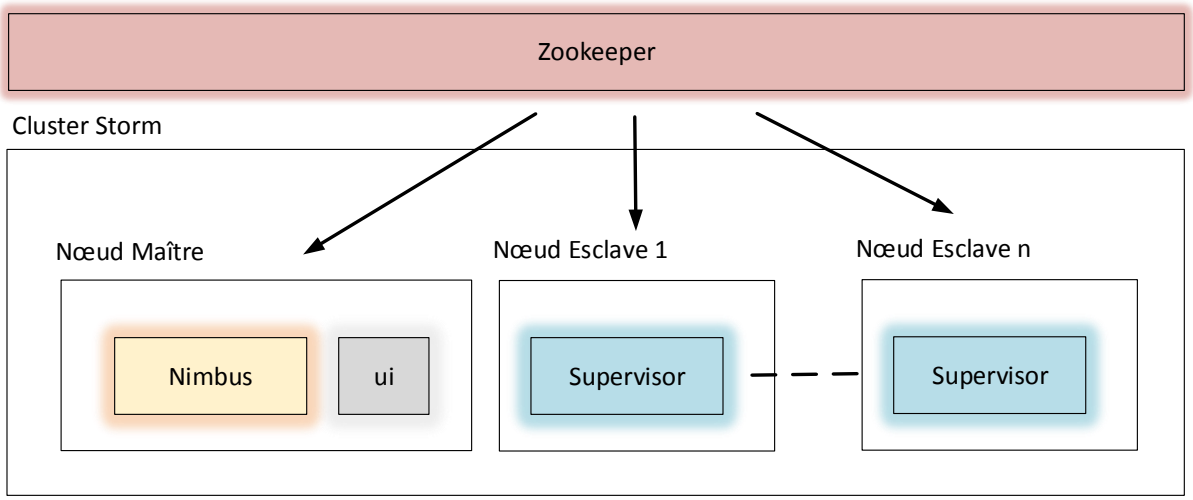

Figure 1. Architecture d'un cluster Storm

Le rôle de chaque élément est le suivant :

- **Zookeeper** : coordonne le fonctionnement de chaque élément ;

- **Nimbus** : affecte les tâches à chaque nœud esclaves ;

- **ui** : fournit une interface graphique de supervision ;

- **Supervisor** : reçoit des tâches de Nimbus, démarre les éléments de la topologie et fournit un résultat.

Dans les tutoriels précédents nous avons hébergé ces éléments sur la même machine physique. Nous allons ici créer un cluster Storm à partir de 3 machines : 1 nœud maître et deux nœuds esclaves.

1. Identifier une machine virtuelle sur le réseau

Il faut disposer d'adresse IP fixes et configurer correctement la machine virtuelle.

Nous allons supposer que nous disposons de 3 adresses de la forme suivante avec un DNS identifié par 172.16.64.250

Address	netmask	gateway
172.16.76.32	255.255.240.0	172.16.79.254
172.16.76.33	255.255.240.0	172.16.79.254
172.16.76.34	255.255.240.0	172.16.79.254

Il faut se rendre dans le menu Paramètres de la VM dans la section Réseau et consulter la configuration de la carte 1. En général, la configuration est positionnée sur NAT, ce qui signifie que la VM peut accéder à Internet mais qu'elle n'est pas accessible sur le réseau.

Il faut basculer le mode d'accès au réseau en "Accès par pont" et sélectionner la carte réseau de la machine hôte (figure 2).

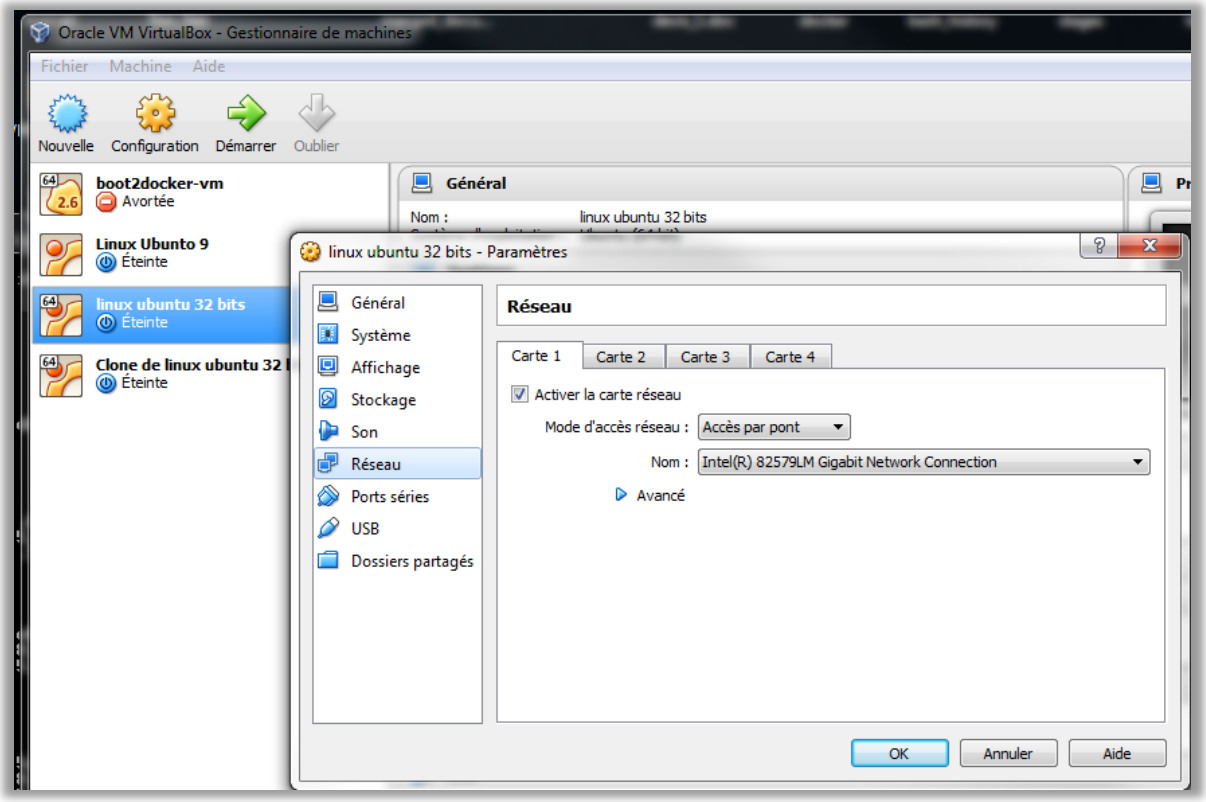

Figure 2. Modification de la configuration réseau

Avec Ubuntu on peut utiliser l'interface graphique pour configurer le réseau. Il suffit de cliquer sur l'icône du haut et de choisir Edit Connections (figure 3).

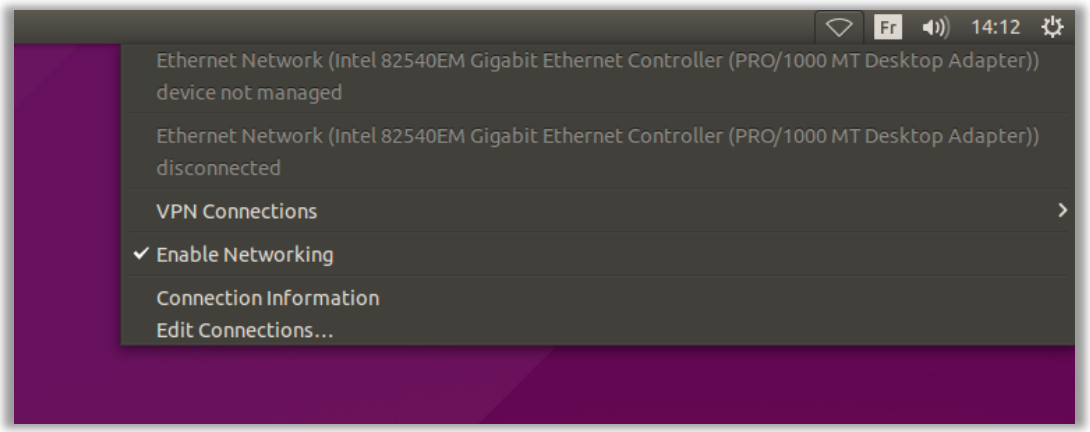

Figure 3. Accès à la configuration réseau sous Ubuntu

L'ensemble des connections apparait et dans la majorité des cas, une seule connexion sera dans la liste (figure 4).

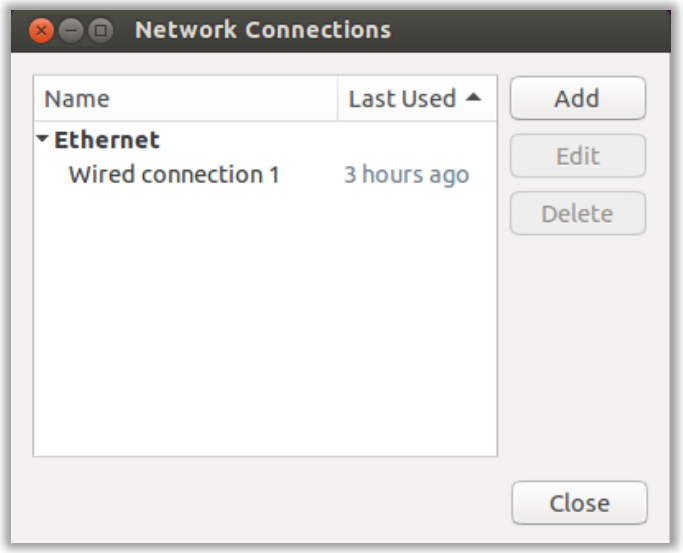

Figure 4. Site de Zookeeper

Il suffit de choisir Edit et de modifier la configuration comme indiqué sur la figure 5.

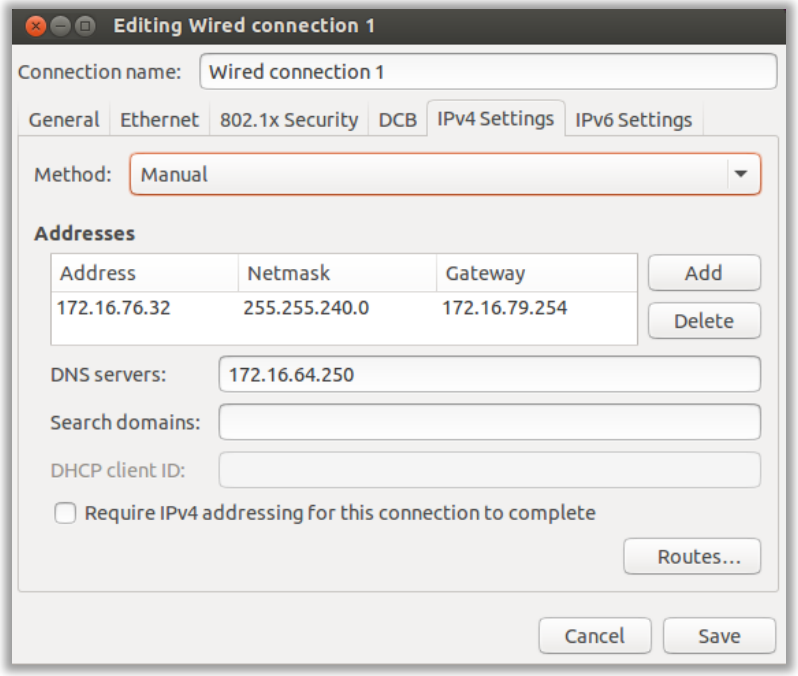

Figure 5. Ajout d'une adresse ip fixe

Une autre méthode, presque aussi rapide, consiste à éditer le fichier "interfaces" via la commande : **sudo gedit /etc/network/interfaces**

Le fichier doit être modifié comme celui de la figure 6.

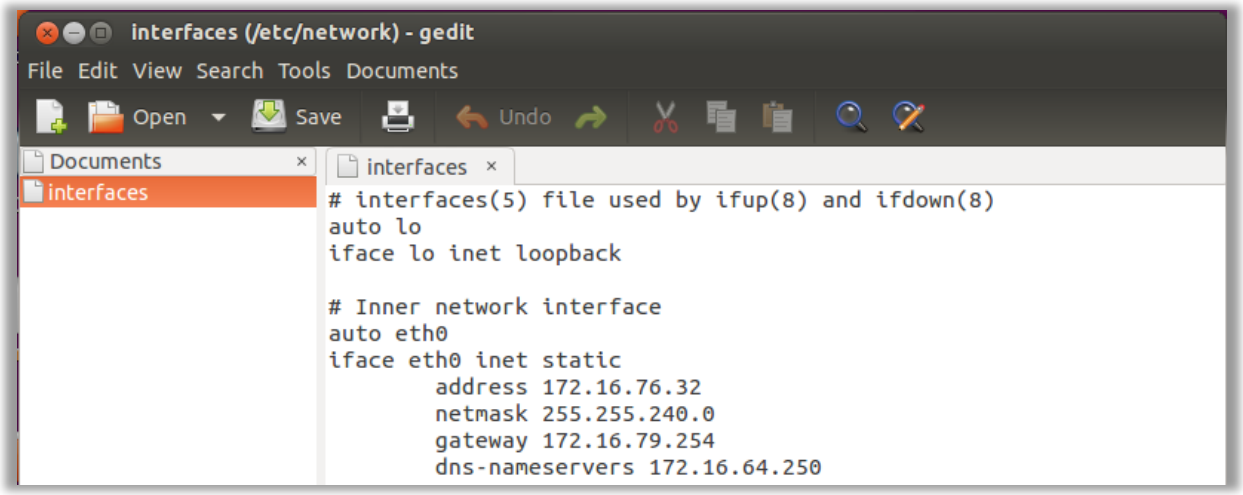

Figure 6. Modification du fichier interfaces

Pour vérifier que tout fonctionne correctement.

1) Il faut démarrer le cluster Storm avec la commande **./lance_storm.sh**
2) Dans un navigateur Internet, on peut consulter le site **localhost:8772** sur la machine virtuelle (figure 7)

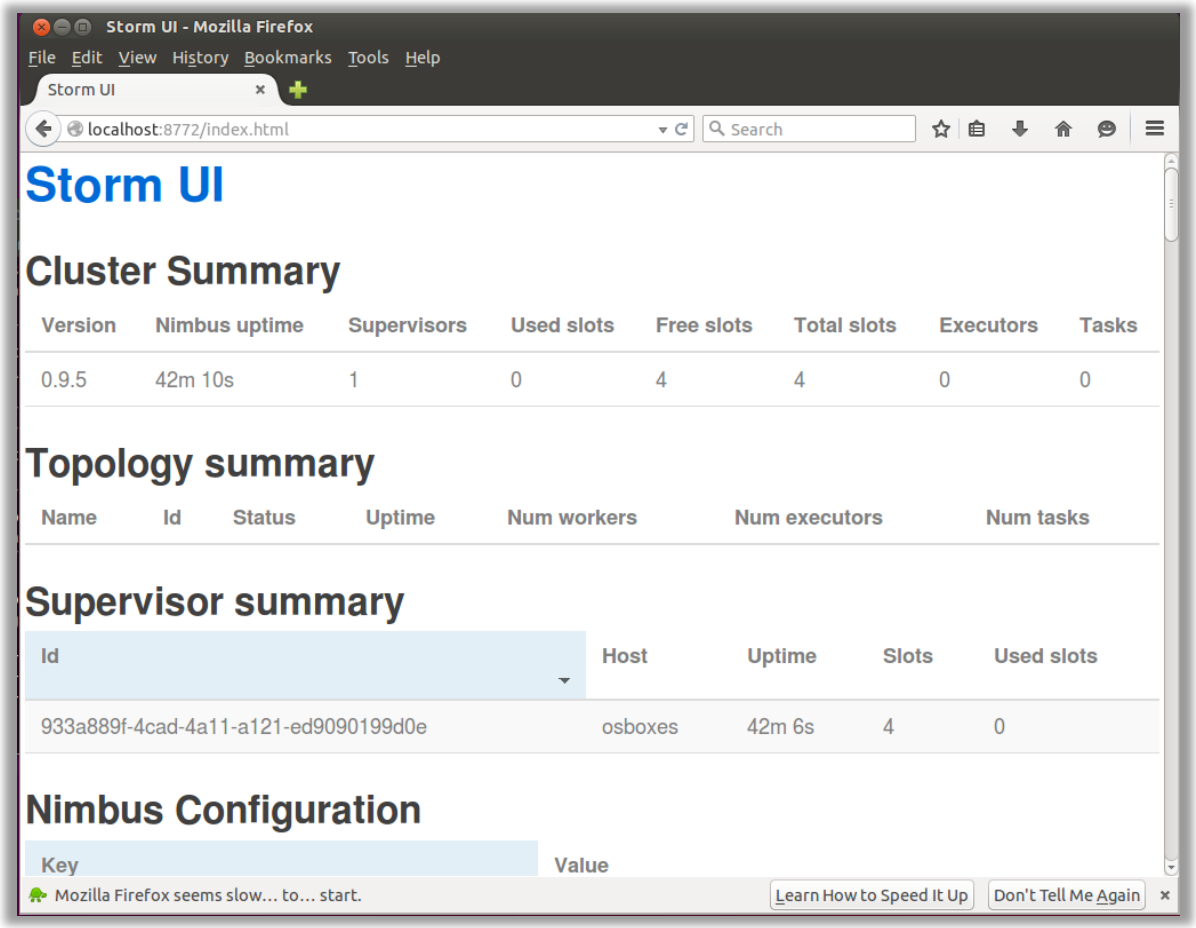

Figure 7. Storm ui consulté à partir de la machine virtuelle

3) Dans un navigateur Internet, on peut consulter le site 172.16.76.32:8772 à partir de la machine hôte (figure 8)

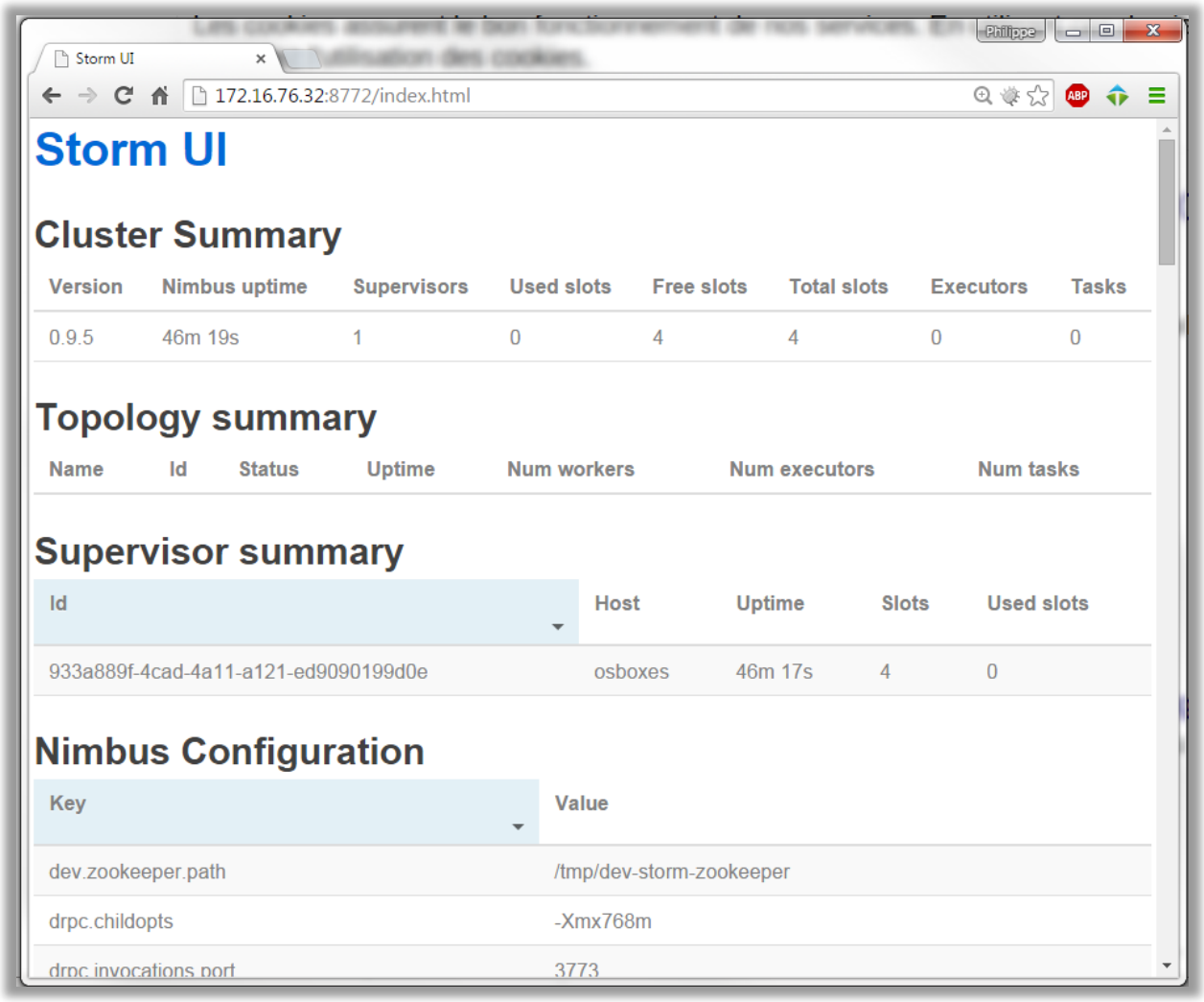

Figure 8. Storm ui consulté à partir de la machine hôte

2. Deux machines dans le cluster Storm

On retient une architecture avec 3 machines virtuelles (figure 9) :

- la machine virtuelle 1 va héberger Nimbus et Zookeeper

- les machines virtuelles 2 et 3 vont héberger chacun un Supervisor

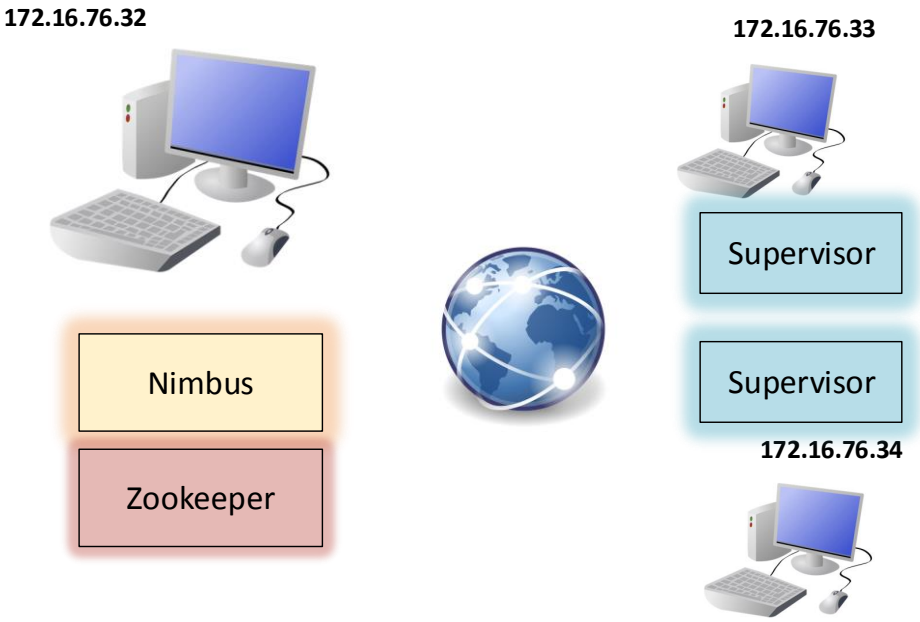

Figure 9. Architecture avec 3 machines virtuelles

Configuration de la machine virtuelle 1

Il faut éditer le fichier **storm.yaml** qui se trouve dans le répertoire

/Desktop/installation/storm/conf

Il faut modifier le contenu en remplaçant **localhost** par **172.16.76.32** (figure 10)

```
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.

############ These MUST be filled in for a storm configuration
storm.zookeeper.servers:
     - "172.16.76.32"
#     - "server2"
#
storm.zookeeper.port: 2181
nimbus.host: "172.16.76.32"
nimbus.thrift.port: 6627
ui.port: 8772
ui.childopts: "-Xmx768m"
storm.local.dir: "/home/osboxes/Desktop/installation/storm/data"
java.library.path : "/usr/lib/jvm/java-7-openjdk-i386/"

supervisor.slots.ports:
   - 6700
   - 6701
   - 6702
   - 6703
```

Figure 10. Le fichier storm.yaml

Sur cette machine, seuls Nimbus et Zookeeper doivent démarrer. Il faut éditer le fichier **lance_storm.sh** et le modifier comme suit :

```
cd /home/osboxes/Desktop/installation/zookeeper-3.4.6/bin/
./zkServer.sh start &
cd /home/osboxes/Desktop/installation/storm/bin/
./storm nimbus &
./storm ui &
```

Configuration de la machine virtuelle 2

Il faut éditer le fichier **storm.yaml** qui se trouve dans le répertoire et le modifier comme cela a été fait pour la machine virtuelle 1. Et cette fois avec l'adresse ip 172.16.76.33.

Sur cette machine, seul le **supervisor** va fonctionner. Il faut éditer le fichier **lance_storm.sh** et le modifier comme suit :

```
cd /home/osboxes/Desktop/installation/storm/bin/
./storm supervisor &
```

Vérification de la configuration

Une fois lancée, sous Windows, on peut se connecter à l'adresse de la machine virtuelle 1 http://172.16.76.32:8772/index.html, sur laquelle est exécutée l'interface de suivi de l'état du cluster Storm (figure 11).

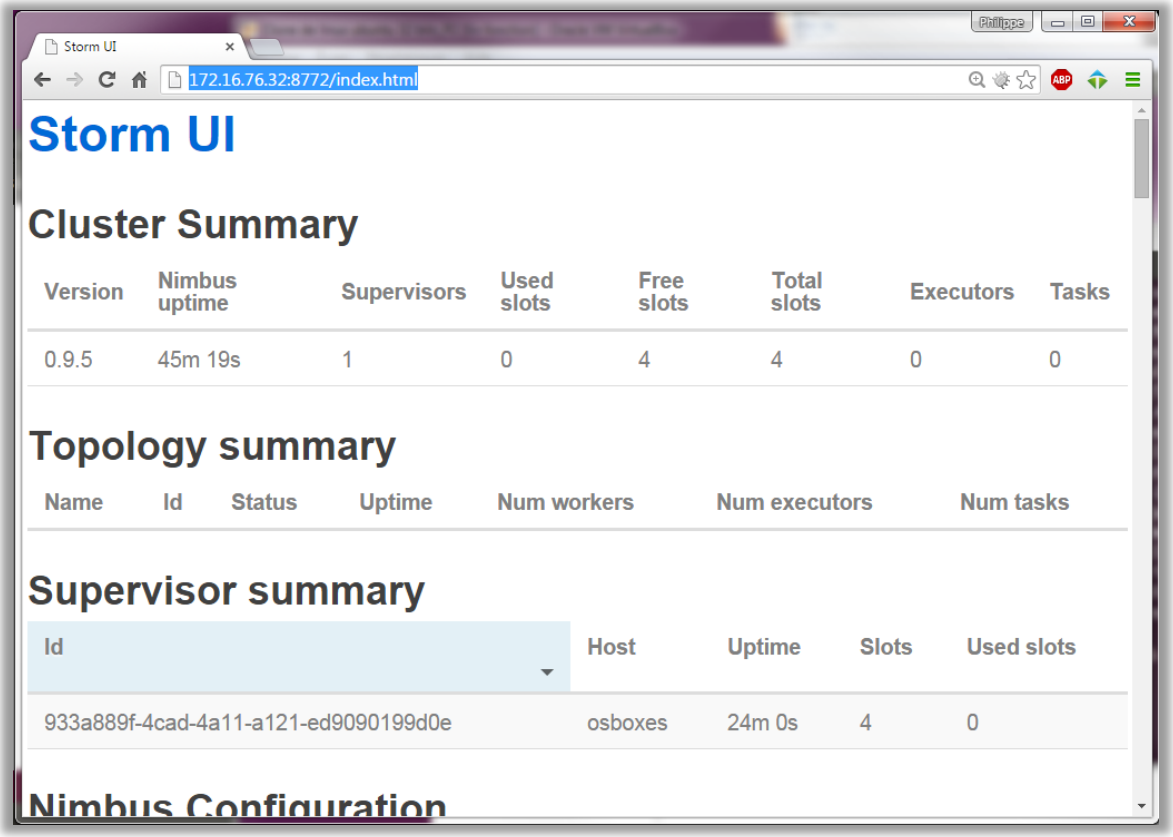

Figure 11. Le cluster Storm

Configuration de la machine virtuelle 3.

Sa configuration est identique à celle de la machine 2 avec l'adresse ip 172.16.76.34.

Vérification de la configuration

La vérification se fait comme pour la machine virtuelle 2 à l'adresse http://172.16.76.32:8772/index.html. Une fois que les machines virtuelle 2 et 3 sont exécutées, l'interface de suivi de l'état du cluster doit afficher ces deux machines.

Remarques : lors de la première connexion d'un nœud (ici la machine virtuelle 2 ou 3) au master (machine virtuelle 1), ils s'échangent des données d'identifications qui sont contenues dans le

dossier "data" du répertoire "storm". Afin d'identifier correctement les nœuds (par exemple lors d'un déplacement physique des machines) il faut penser à supprimer ce dossier avant une nouvelle connexion si on souhaite réinitialiser l'identification des machines.

3. Tests

Il faut se rendre sur la machine Maître dans le répertoire **bin** de **storm** et taper en ligne de commande :

```
storm jar DemoStorm.jar demostorm.DemoStorm DemoStorm demostorm.DemoStorm DemoStorm
-c nimbus.host=localhost
```

Si on prend une des "slave machine", dans le répertoire /Storm/logs, on peut trouver les fichiers **worker-xxxx.log** (figure 12).

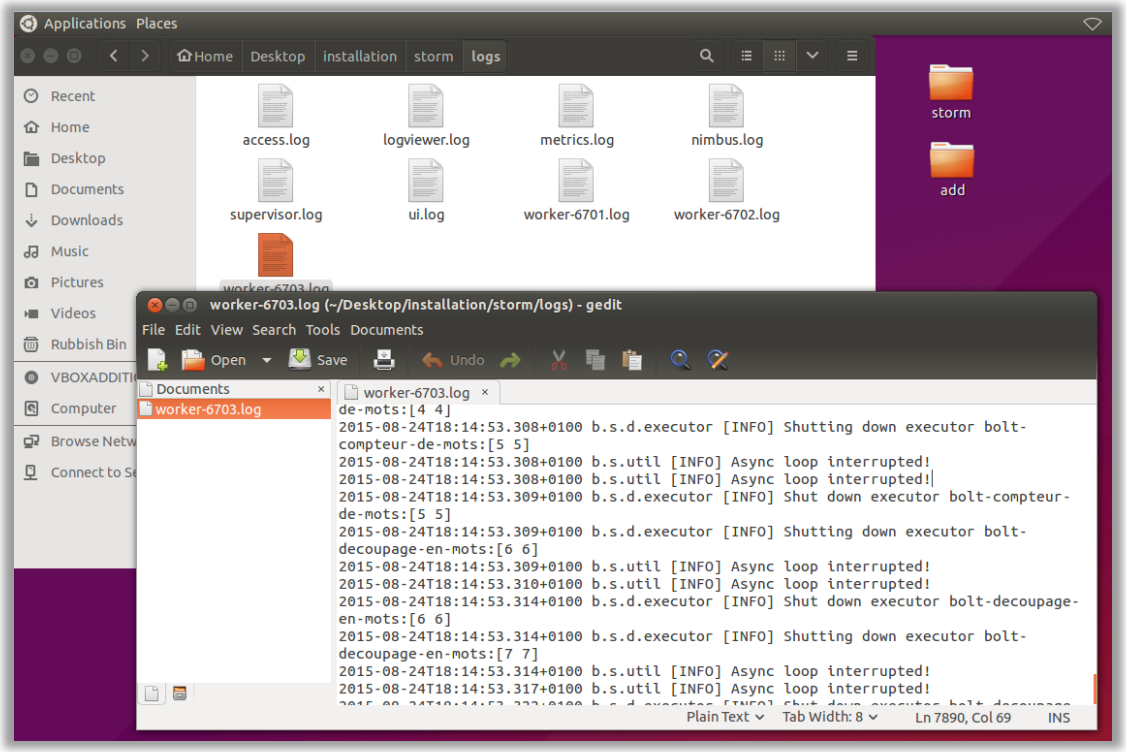

Figure 12. Un job exécuté sur l'esclave 1

On peut parcourir le fichier et on trouvera dans le fichier, une partie des affichages qu'on ne trouve pas sur la machine Maître (figure 13).

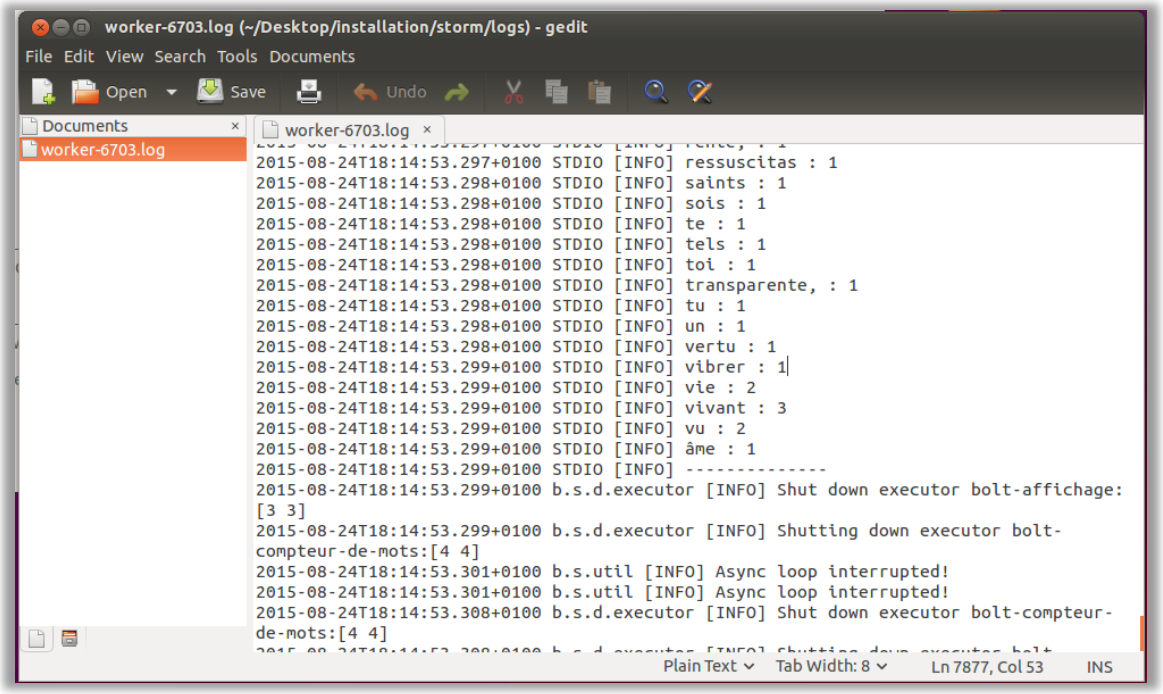

Figure 13. Une partie des résultats dans le fichier log.

4. Principes d'exécution en parallèle

Comme nous l'avons rappelé au début de ce document, un cluster Storm met en jeu :

- **Zookeeper** : coordonne le fonctionnement de chaque élément ;

- **Nimbus** : affecte les tâches à chaque nœud esclaves ;

- **ui** : fournit une interface graphique de supervision ;

- **Supervisor** : reçoit des tâches de Nimbus, démarre les éléments de la topologie et fournit un résultat.

L'ensemble de cette architecture s'exécute sur des ressources matérielles (machines) qu'on appelle des nœuds : un nœud maître et plusieurs nœuds esclaves.

On peut retenir les définitions suivantes :

- Un master est une machine physique sur laquelle est installée un ensemble d'outils (Zookeeper, Storm…) afin de gérer le cluster.

- Un nœud (**node**) est une machine virtuelle qui exécute une partie de la topologie (ainsi une machine physique peut contenir plusieurs nœuds de machines virtuelles visible à partir du master.

- Un **worker** (travail à faire) est un processus Java affecté à une JVM sur un nœud. Plusieurs workers peuvent être affectés à un même nœud. Un worker peut contenir plusieurs **executors**.

- Un **executor** est un thread c'est-à-dire un ensemble d'opérations exécutées de manière séquentielle. Un executor peut contenir plusieurs **tasks**.

- Une tâche (ou **task**) peut être considérée comme l'instanciation d'un Spout ou d'un Bold. Dans l'architecture Storm, c'est donc l'élément le plus petit qui réalise une fonction.

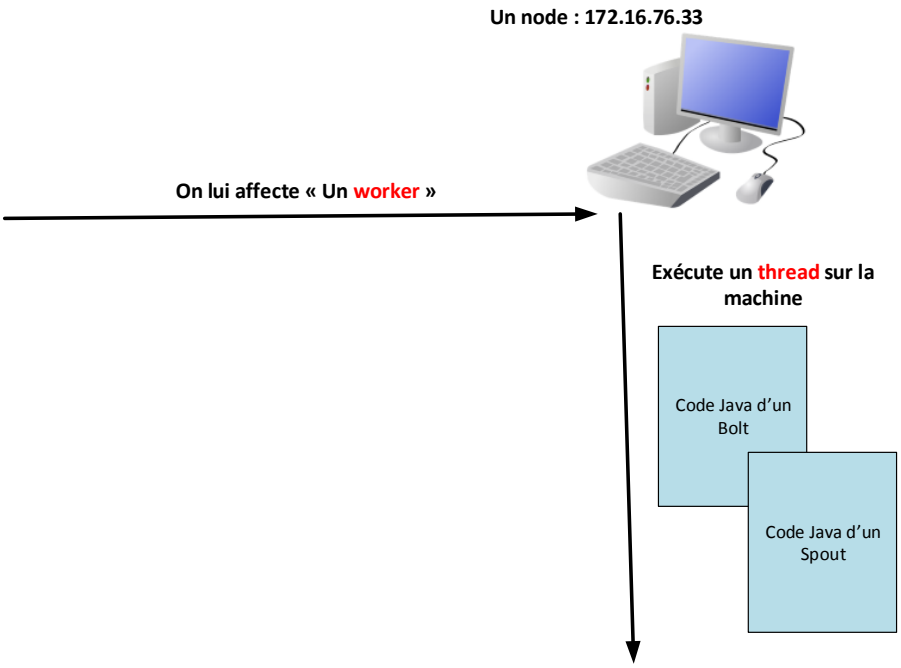

Figure 14. Principe des threads et worker

Ce principe appliqué au problème de décompte des mots donne le schéma de la figure 15. On met en évidence que 4 threads s'exécutent en parallèle dans le système, la communication se faisant par l'intermédiaire des streams échangés entre les threads.

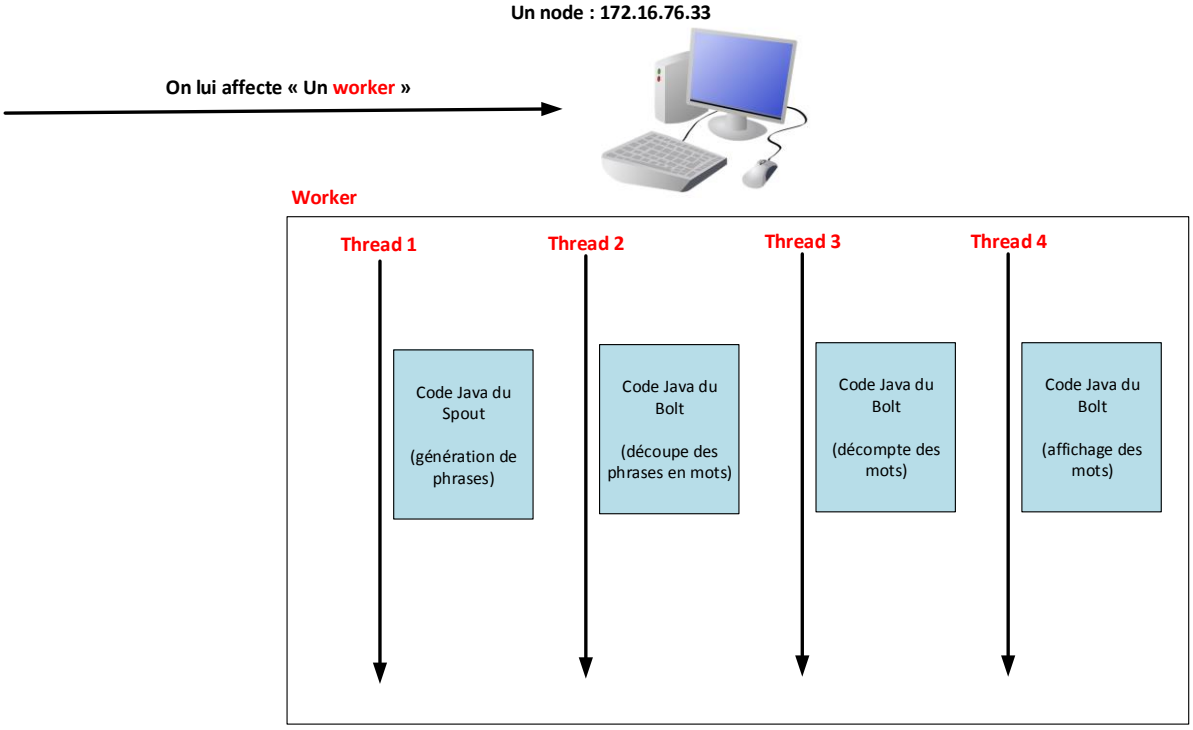

Figure 14. Principe des threads pour le décompte des mots

Remarque :

L'exécution en parallèle se fait uniquement au niveau des threads. Chaque tâche (spout ou bolt) est exécutée dans un thread.

5. Principes de gestions des Threads

Définir le nombre de workers

Par défaut le nombre de worker est fixé à 1, mais on peut spécifier le nombre de worker en utilisant la méthode setNumWorker(). Ce qui donne un code comme celui-ci :

```
Config configuration = new Config();
configuration.setDebug(true);
configuration.setNumWorkers(2);
```

Avec cette modification, le code Java du main() devient :

```
Config configuration = new Config();
configuration.setDebug(true);
configuration.setNumWorkers(2);

if (args != null && args.length > 0) {
  StormSubmitter.submitTopology(args[0],
                                configuration,
                                topologie.createTopology());
}
else
  {
  LocalCluster cluster = new LocalCluster();
  cluster.submitTopology(TOPOLOGY_NOM,
                         configuration,
                         topologie.createTopology());
  int i=1;
  while (spout_genere_phrases.consulter_etat()==0)
  {
   Utils.waitForSeconds(1);
  }
  Utils.waitForSeconds(20);

  cluster.killTopology(TOPOLOGY_NOM);
  cluster.shutdown();
  }
```

Configurer les tâches et les threads : ajout d'un spout

Le nombre de threads à utiliser pour une tâches (bolt ou spout) est défini au moment où on définit les Streams qui relient les composants.

Pour ajouter deux spouts de type "spout_genere_phrases" il faut ajouter un troisième paramètres de valeur 2 :

```
topologie.setSpout(SPOUT_ID, spout_genere_phrases,2);
```

Par défaut, cela crée en tout 2 tasks de "spout_genere_phrases" dans 2 executors différents. Dans ce cas, la topologie devient celle de la figure 16.

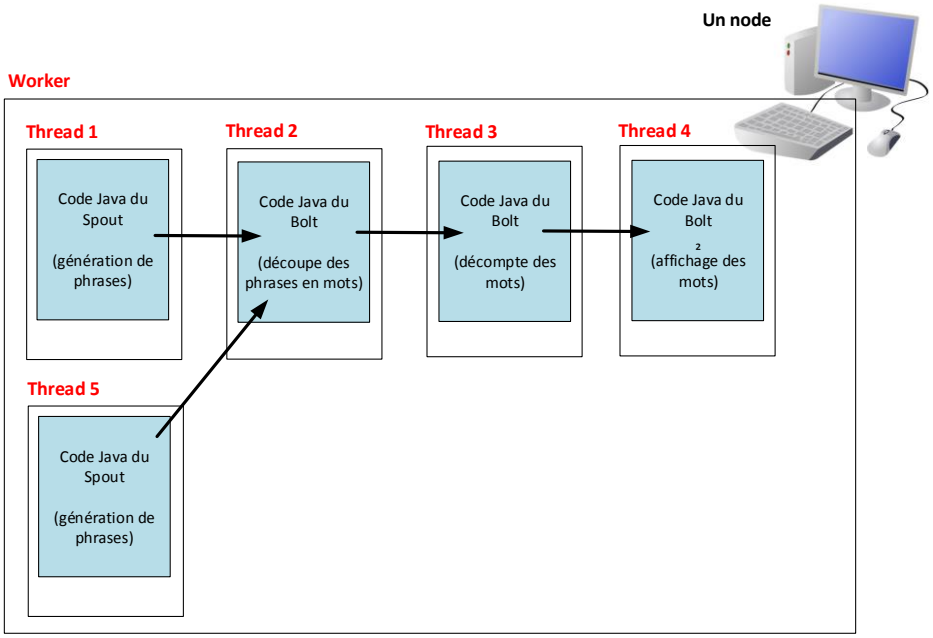

Figure 16. Principe des threads pour le décompte des mots

Configurer les tâches et les threads : ajout d'un bolt de découpage

A l'intérieur d'un même thread, on peut exécuter plusieurs bolts et on obtient ainsi une topologie dans laquelle le thread 2 va exécuter 4 tâches de type "bolt_decoupe".

```
topologie.setBolt(BOLT_DECOUPE_ID,
                  bolt_decoupe, 2).
                   setNumTasks(4).shuffleGrouping(SPOUT_ID);.
```

Ce code crée donc 4 tasks de type "bolt_decoupe" qui sont exécutés dans 2 executors différents. Storm équilibre automatiquement : il y a donc 2 tasks dans chacun des executors. L'ensemble de ces tasks reçoivent un flux du Spout. La topologie est alors celle de la figure 17.

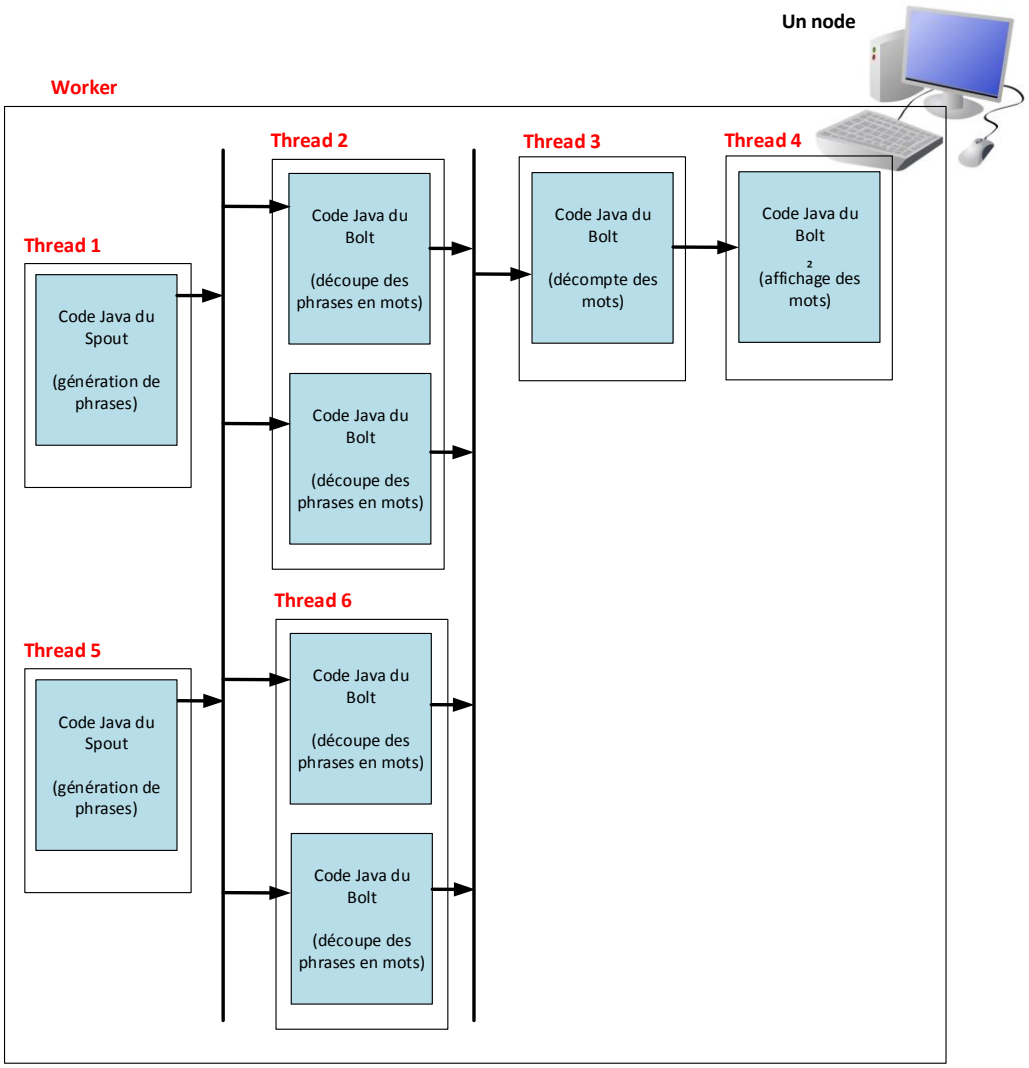

Figure 17. Principe des threads pour la découpe des mots

Configurer les tâches et les threads : ajout d'un bolt de comptage

Ainsi à l'intérieur d'un même executor, on peut exécuter plusieurs tasks. On réalise cela pour le bold "bolt_compte". On obtient une topologie dans laquelle le thread 3 va exécuter 2 bolts.

```
topologie.setBolt(BOLT_COMPTEUR_ID,
                  bolt_compte,1).setNumTasks(2)
                  fieldsGrouping(BOLT_DECOUPE_ID, new Fields("word"));
```

En conclusion, la topologie est celle de la figure 18, dans l'hypothèse où il n'y aurait qu'un seul worker.

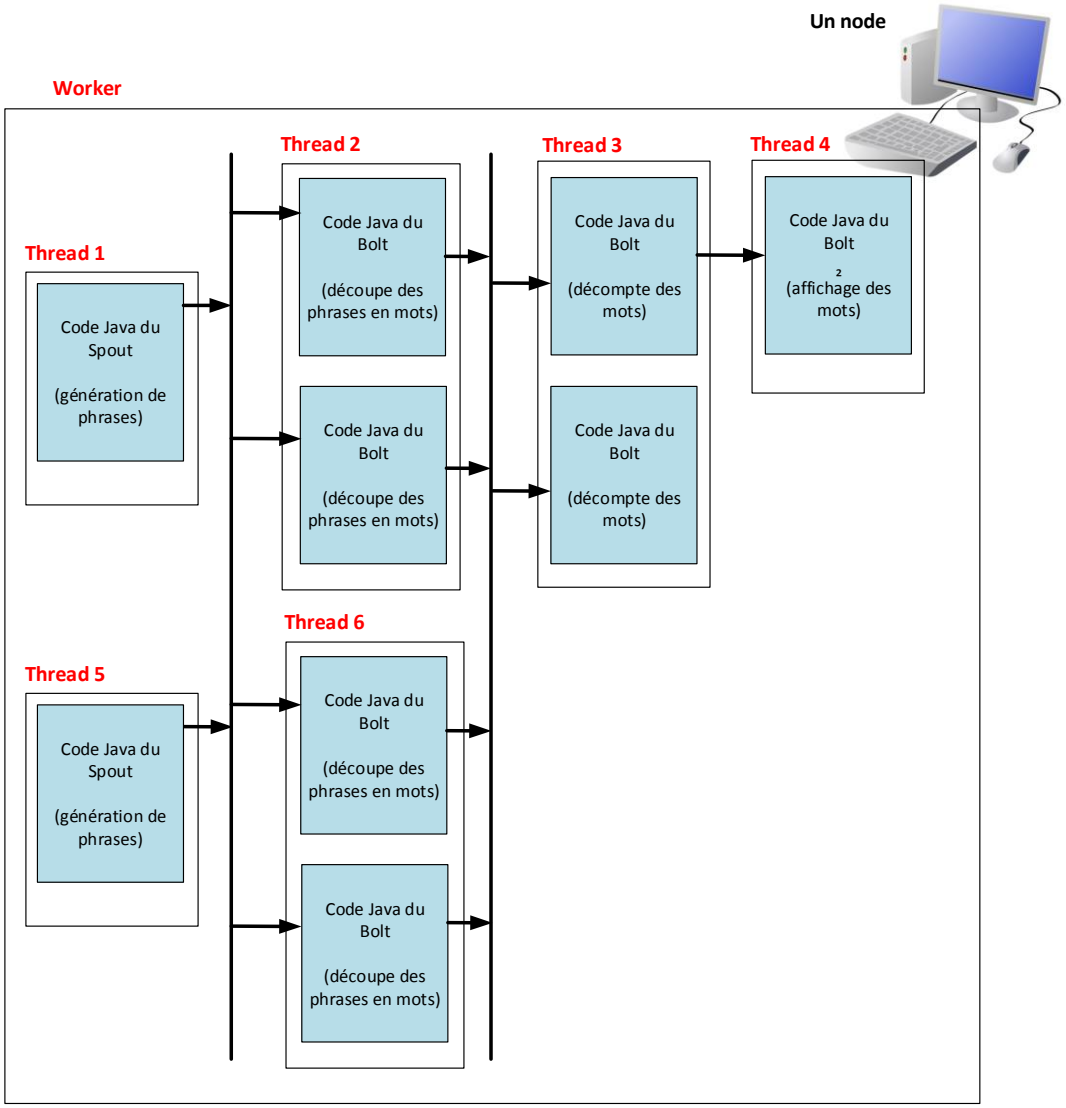

Figure 18. Principe des threads pour le décompte des mots

Or nous avons choisie d'utiliser 2 workers : Storm va donc équilibré cette topologie sur 2 nœuds différents. Il est probable que la topologie avec 2 workers soit celle de la figure 19.

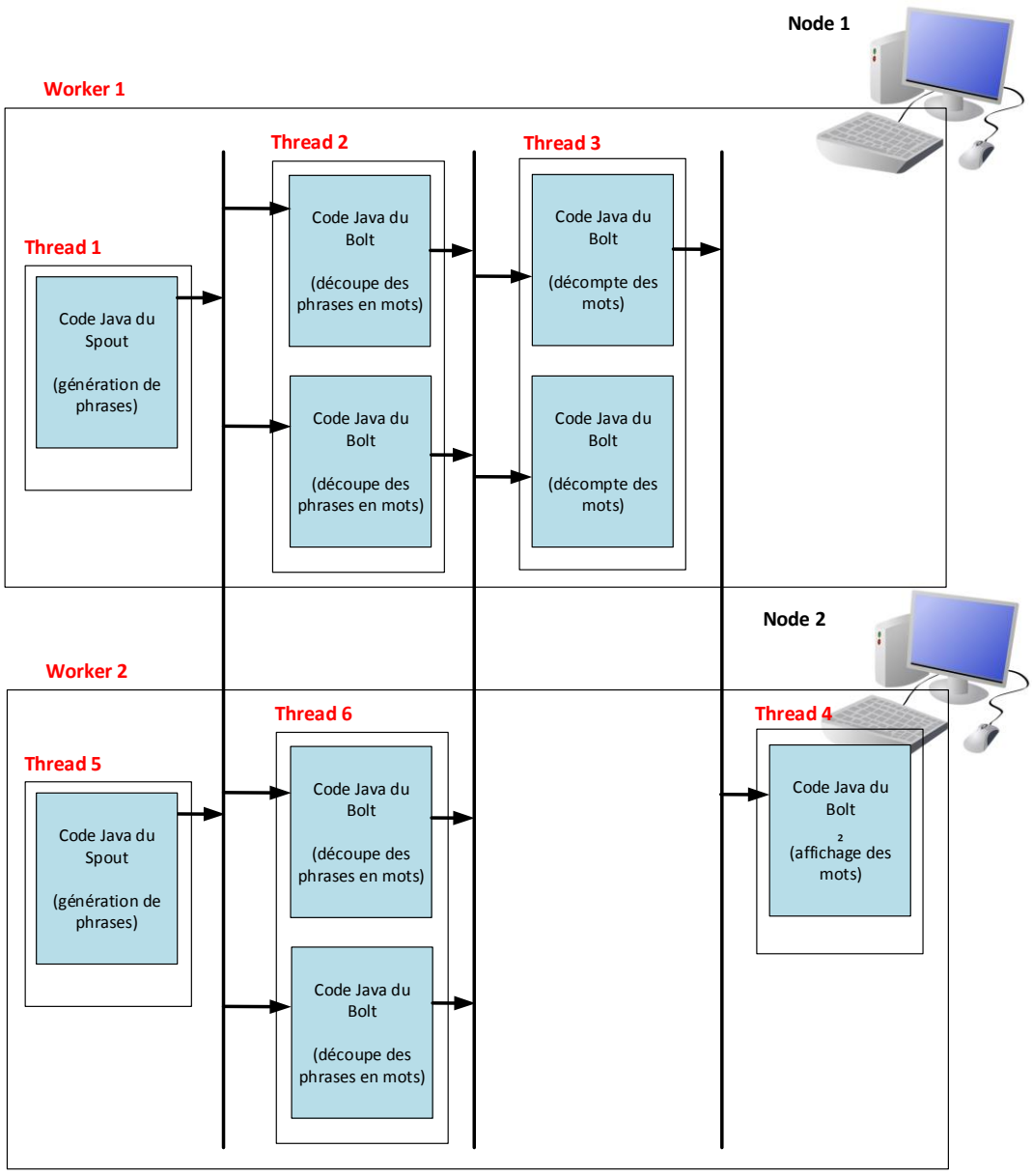

Figure 19. Définition de la topologie avec 2 workers

6. Principes de gestion des streams

Il a 7 politiques de gestion des streams.

Regroupements	Description
Shuffle grouping	Affection des éléments de manière aléatoire entre les destinations en équilibrant les flux
Fields grouping	Il est définit en donnant un nom de champ (par exemple "word" dans l'exemple du tutoriel 1). Pour une valeur de champ donnée, le flux de la source vers la même destination
All grouping	Chaque élément du flux est dupliqué à destination de tous les nœuds destination
Global grouping	Un élément du flux est routé vers la destination ayant le plus petit ID

None grouping	**= Shuffle grouping**
Direct Grouping	Un élément de flux est envoyé à un Bolt particulier en utilisant la méthode emitDirect(). Ceci ne peut s'appliquer qu'au stream déclaré de type direct.
Local or shuffle grouping	Produit une gestion similaire à Suffle Grouping mais en affectant le stram aux bolts tournant sur le même tread. Si un bolt n'existe pas dans le même thread, on applique Shuffle Gouping.

Table 1. Les différentes politiques de gestion des streams

Il est possible de définir sa propre politique de gestion des streams en utilisant l'interface **CustomStreamGrouping** qui se présente comme suit :

```
public interface CustomStreamGrouping extends Serializable {
   void prepare(WorkerTopologyContext context,
                GlobalStreamId stream,
                List<Integer> targetTasks);

   List<Integer> chooseTasks(int taskId, List<Object> values);
}
```

7. Test : 3 workers, 3 ports par worker

Lors de la configuration des nœuds et du master plusieurs paramètres sont à définir dans le fichier storm.yalm. Pour ce test, il s'agit de configurer un master (avec cinq ports ouverts : 6700, 6701 et 6702, 6703 et 6704) et 3 nœuds (appelés workers) qui se connectent tous les trois aux ports 6701, 6702 et 6703. La figure 20 résume cette configuration.

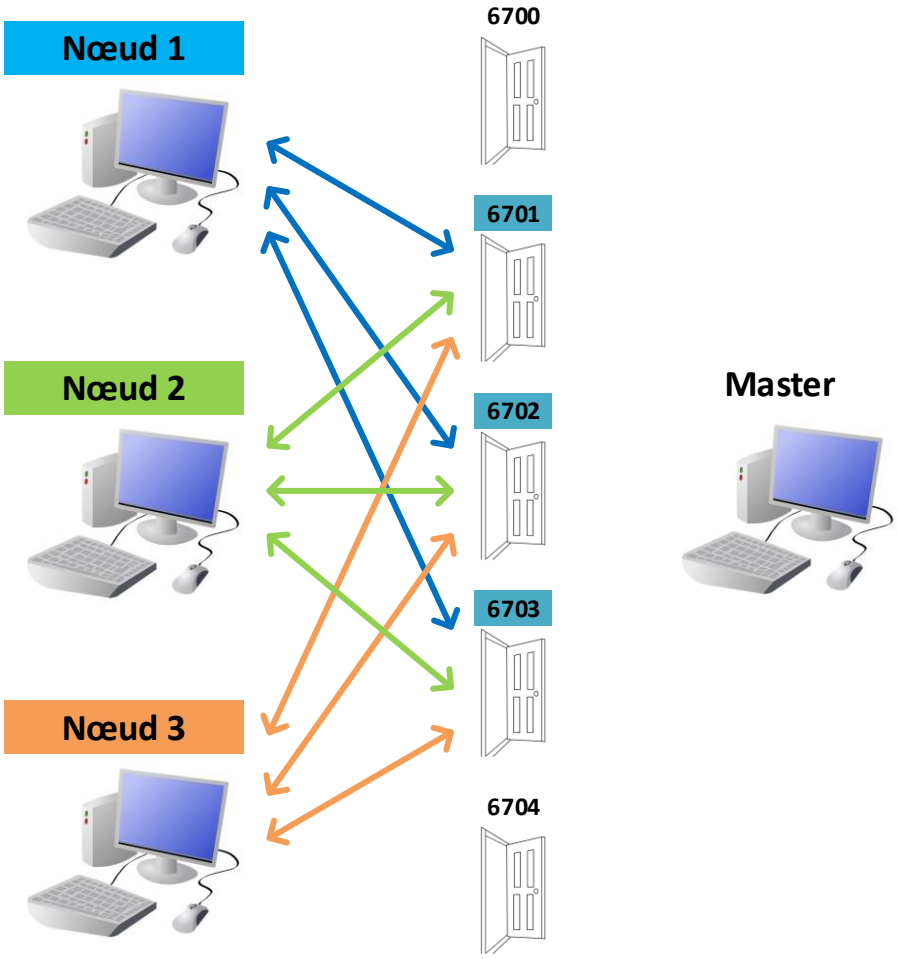

Figure 20. Topologie du test 3 workers, 3 ports par worker

Une topologie avec cinq bolts indépendants est envoyée au master. Le code représente en réalité cinq fonctions de calculs complexes qui ont été manipulées pour durer plusieurs secondes : cela permet de visualiser les temps de calculs dans les fichiers logs. Les résultats sont donnés sous forme de diagramme de Gantt à la figure 21.

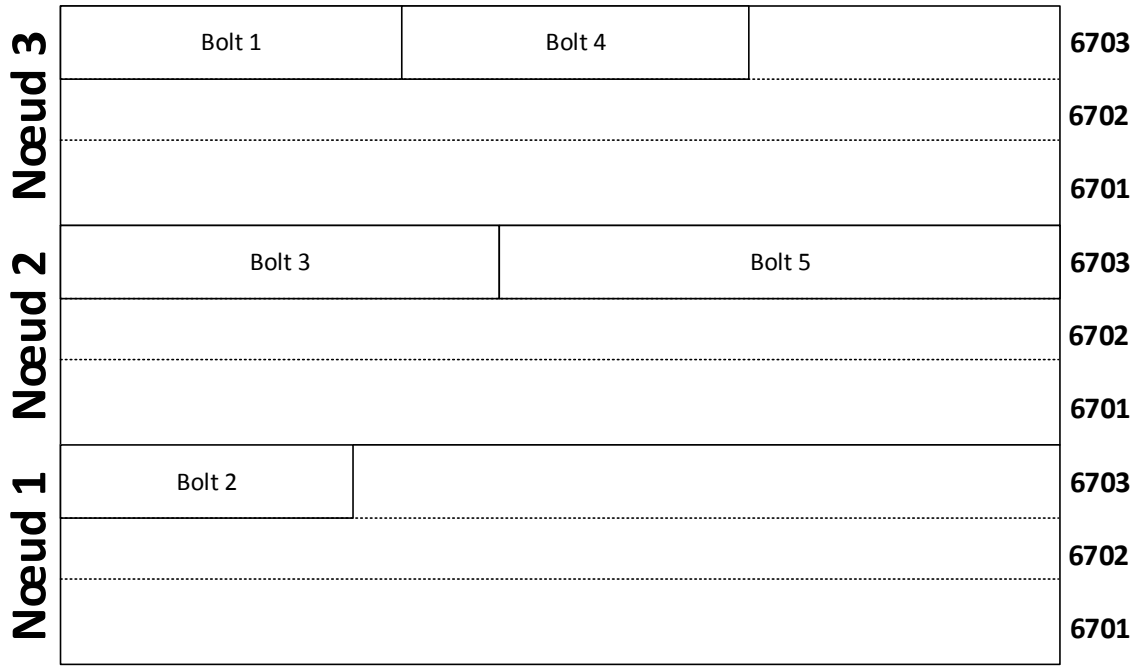

Figure 21. Résultat de la topologie du test 3 workers, 3 ports par worker

8. Test : 3 workers, 1 ports par worker

Lors de la configuration des noeuds et du master plusieurs paramètres sont à définir dans le fichier storm.yalm. Pour ce test, il s'agit de configurer un master (avec cinq ports ouverts : 6700, 6701 et 6702, 6703 et 6704) et 3 nœuds (appelés workers) qui se connectent chacun à un port spécifique : nœud 1 avec le port 6701, nœud 2 avec le port 6702 et nœud 3 avec le port 6703. La figure 22 résume cette configuration.

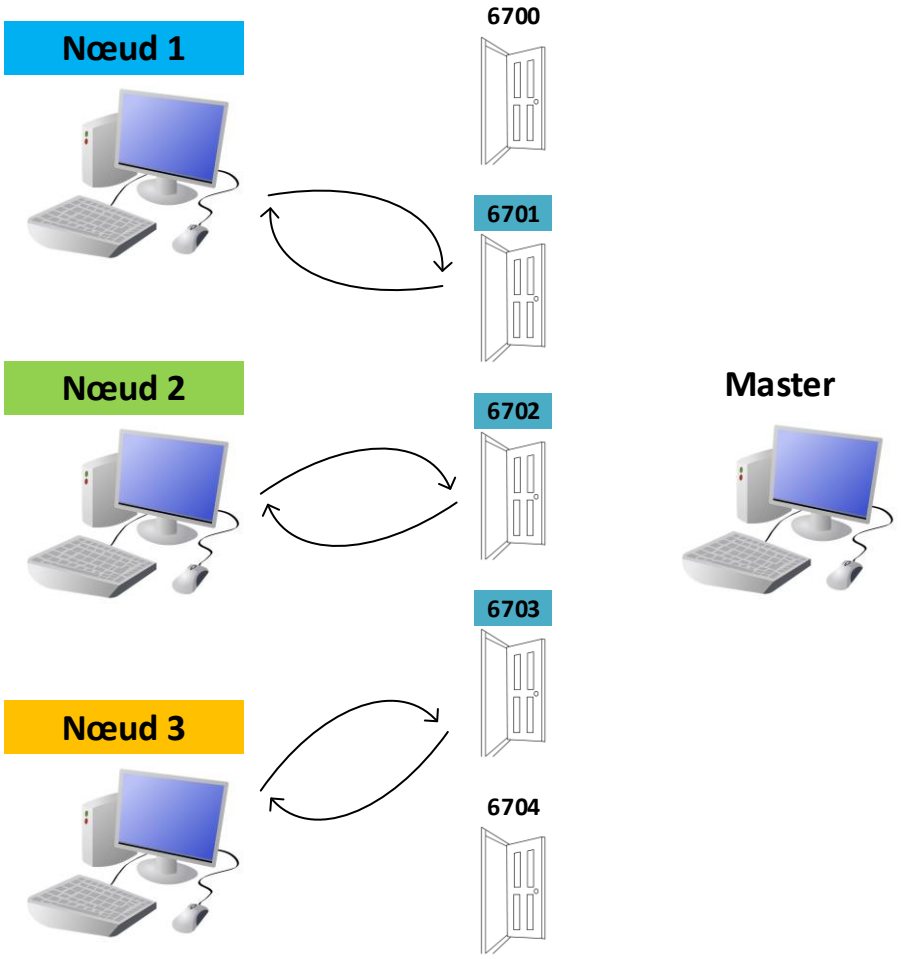

Figure 22. Topologie du test 3 workers, 1 ports par worker

La même topologie avec cinq bolts indépendants est envoyée au master. Les résultats sont donnés sous forme de diagramme de Gantt à la figure 21.

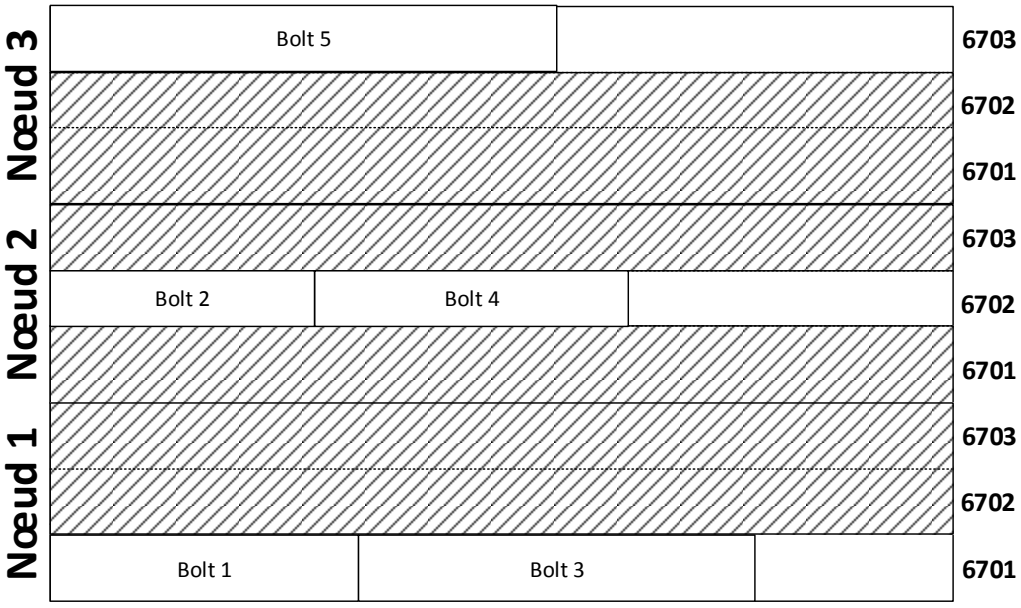

Figure 21. Résultat de la topologie du test 3 workers, 1 ports par worker