# M2 TIW6 - TP Vélocité
# Traitement de flux via *Apache STORM*

UCBL - Département Informatique de Lyon 1 – 2016

L'objectif de ce TP est de vous familiariser avec les topologies STORM permettant de traiter des flux de données et de vous sensibiliser aux problèmes de congestions d'opérateurs qui peuvent apparaître en fonction de l'évolution du débit du flux en entrée.

**Modalités de rendu** : L'export de votre projet maven sera mis dans une archive .zip que vous déposerez sur TOMUSS avant le **16/12/2016** à 23h.

## Première partie

# Prise en main d'Apache STORM

## 1 Mise en place d'une topologie

Cette première partie vous permet de pendre en main l'application Apache STORM en définissant des topologies pour traiter un flux de données à débit constant correspondant aux coordonnées d'un pion dans un espace de jeu.

### 1.1 Environnement

Vous trouverez la documentation sur Apache STORM à l'adresse :
http://storm.apache.org/releases/1.0.2/index.html

Dans ce TP vous disposez d'une VM sur le cloud du département dont l'adresse IP qui vous a été attribué en début de séance et spécifiée sur TOMUSS.

Pour vous connecter à votre VM vous utiliserez une connexion *ssh* avec la clé *pedabdcloud* qui vous a été transmis par mail :

```
ssh -i <keyPath>/pedabdcloud -X ubuntu@192.168.73.x
```

Comme il va être nécessaire de lancer plusieurs applications, il est conseillé d'utiliser *screen*[1] pour le lancement de STORM et d'éviter un nombre trop important de terminaux ouvert en *ssh*.

Dans le répertoire */home/ubuntu* de votre VM, vous avez :

- Pour l'émission du flux de coordonnées :
  - le script *startStream.sh* qui émet le flux de coordonnées sur le port 9001
  - le script *stopStream.sh* qui permet de stopper proprement votre flux des coordonnées.
- Pour le traitement de flux :
  - le répertoire *./lib/zookeeper-3.3.6* contenant les fichiers de l'application de monitoring ZOOKEEPER.
  - le répertoire *./lib/apache-storm-1.0.2* contenant les fichiers de l'application de Apache STORM.
  - le programme *./stormTP-0.1-with-dependencies.jar* contenant une topologie qui sera exécutée par*Apache STROM* (c'est ce jar que vous aurait à générer pour soumettre une topologie).
  - l'archive *Storm.zip* contenant le projet maven avec les classes JAVA qui vous permettront de définir vos topologies *Apache STROM* pour traiter les flux.
- Pour le partage de ressources (CPU, RAM) avec un tiers :
  - le répertoire *./forOther/apache-storm-1.0.2* contenant les fichiers de l'application de Apache STORM utilisable par d'autres que vous.
- Pour le monitoring du flux de sortie :
  - le script *main.js* qui permet de lancer une application NODE.JS pour écouter du flux de sortie (sur le port 9002) généré par votre topologie STORM.

En local sur votre machine, vous pouvez récupérer l'archive *Storm.zip*. A partir de cette archive, vous pouvez importer le projet MAVEN dans votre IDE Eclipse. Le *pom.xml* de votre projet est configuré pour générer une topologie STORM dans le programme *stormTP-0.1-with-dependencies.jar* que vous trouverez dans le répertoire *target*. Après toute modification de vos classes définissant votre topologie, il sera nécessaire de générer le *stormTP-0.1-with-dependencies.jar* via la commande *'mvn assembly :assembly'* puis de transférer le .jar générée sur votre VM (via *scp*).

1. https://doc.ubuntu-fr.org/screen

## 1.2 Paramètres de configuration

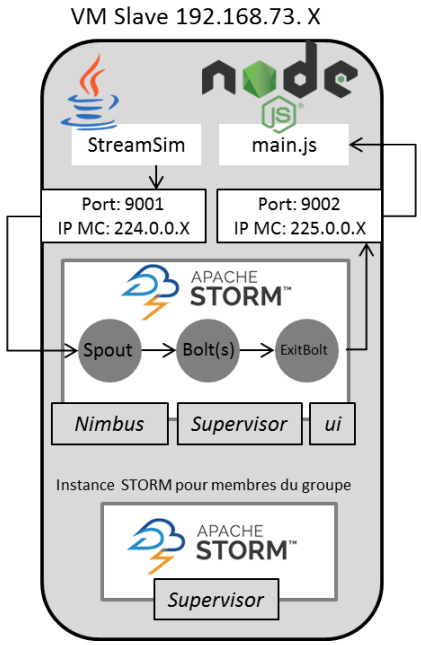


FIGURE 1 – Architecture globale

| Groupe | IP VMs (192.168.73.*X*) |
|---|---|
| A | X= 205...209 |
| B | X= 210...214 |
| C | X= 215...219 |
| D | X= 220...224 |
| E | X= 225...229 |
| F | X= 230...234 |
| G | X= 235...239 |
| H | X= 240...245 |

FIGURE 2 – Affectation des groupes

### 1.2.1 Ports et IP multicast

Comme le montre la Figure 1 , pour pouvoir configurer votre VM, vous avez besoin :

— de l'adresse IP de votre VM

— de l'adresse IP multicast d'émission du flux d'entrée sur le port 9001

— de l'adresse IP multicast d'émission du flux de sortie sur le port 9002

L'IP de votre VM est de la forme **192.168.73.X** avec un X allant de 205 à 245. Par conséquent, votre adresse IP multicast d'émission du flux d'entrée est **224.0.0.X** et votre adresse IP multicast d'émission du flux de sortie est **225.0.0.X**.

Ainsi, si votre adresse IP est *192.168.73.230*, votre adresse IP multicast d'émission du flux d'entrée est *224.0.0.230* et votre adresse IP multicast d'émission du flux de sortie est **225.0.0.230**.

### 1.2.2 STORM

En ce qui concerne la configuration de STORM, elle est spécifiée dans le fichier *./lib/apache-storm-1.0.2/conf/storm.yaml*.

### 1.2.3 Groupes

Pour réaliser certains exercices, vous êtes amenés à consulter les flux d'autres participants. Pour cela, nous allons constituer des groupes comme le spécifie le tableau de la Figure 2.

## 1.3 Lancement des applications

### 1.3.1 Lancement du générateur de flux

Exécuter :

```
./startStream.sh <nomsBinomes> <groupe> <ipMulticastIN>
```

où *<nomsBinomes>* correspond à une chaîne de caractères vous identifiant (concaténation de vos noms ou pseudo), *<groupe>* correspond au nom de votre groupe (cf liste des groupes en Figure 2) et *<ipMulticastIN>* correspond à votre adresse IP multicast d'émission du flux.

### 1.3.2 Lancement de STORM

— Dans le répertoire *./lib/zookeeper-3.3.6/bin*, exécuter :

```
./zkServer.sh start
```

— Dans le répertoire *./lib/apache-storm-1.0.2/bin*,

- pour lancer votre master Nimbus, exécuter :

```
./storm nimbus
```

- pour lancer votre interface de monitoring de Storm [2] sur la page *http ://<votreIP> :8080*, exécuter :

```
./storm ui
```

- pour lancer un nœud de traitement sur votre VM, exécuter :

```
./storm supervisor
```

### 1.3.3 Lancement de votre topologie

Comme indiqué précédemment, vous avez une topologie qui vous est proposée dans le fichier /home/ubuntu/stormTP-0.1-with-dependencies.jar. Cette topologie est composée de 3 opérateurs :

— *StreamSimSpout* qui écoute le port 9001 sur votre IP multicast du flux d'entrée et bufferise les messages UDP reçus.

— *TestStatelessBolt* qui ne fait que transmettre le message reçu à l'opérateur suivant.

— *ExitBolt* qui émet à l'adresse IP multicast du flux de sortie sur le port 9002 les messages reçus.

Pour lancer cette topologie, exécuter dans le répertoire *./lib/apache-storm-1.0.2/bin* :

```
./storm jar /home/ubuntu/stormTP-0.1-with-dependencies.jar
      stormTP.topology.TopologyT1 <X>
```

où *<X>* correspond au dernier nombre de votre adresse IP.
**Remarque** : Vous disposez dans le package *stormTP.operator.test* de quatre exemples de classes implémentant respectivement un opérateur *stateless*, *stateful*, *stateless* avec fenêtrage, *stateful* avec fenêtrage.

### 1.3.4 Lancement du monitoring

Pour vous assurer que tout se passe correctement, vous disposez de deux moniteurs.
Pour pouvoir visualiser les tuples en sortie de votre topologie, exécuter dans votre *home* :

```
node main.js <ipMulticastOUT>
```

où *<ipMulticastOUT>* correspond à votre IP multicast du flux de sortie.

Pour visualiser le trafic dans votre topologie, vous avez l'interface *ui* comme représenté sur la Figure 3 et la Figure 4.

Sur la page d'accueil et dans la section "Topology summary", vous devez avoir *topoT1* qui apparaît. En cliquant sur *topoT1*, vous accédez à l'interface dédiée à cette topologie représentée sur la Figure 4. Les deux boutons importants dans cette interface, sont les boutons :

— **Kill** : qui vous permet d'arrêter votre topologie.

— **Show visualization** : qui permet de visualiser graphiquement votre topologie.

2. http://www.malinga.me/reading-and-understanding-the-storm-ui-storm-ui-explained/

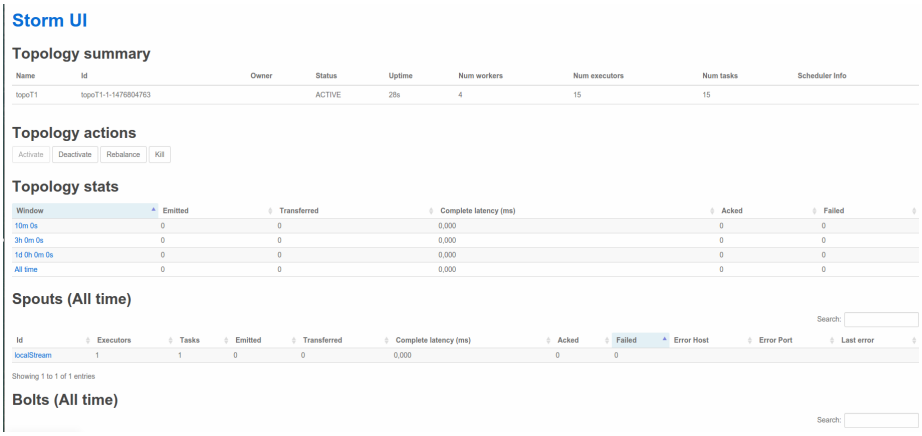


FIGURE 3 – Interface Storm UI : Page d'accueil

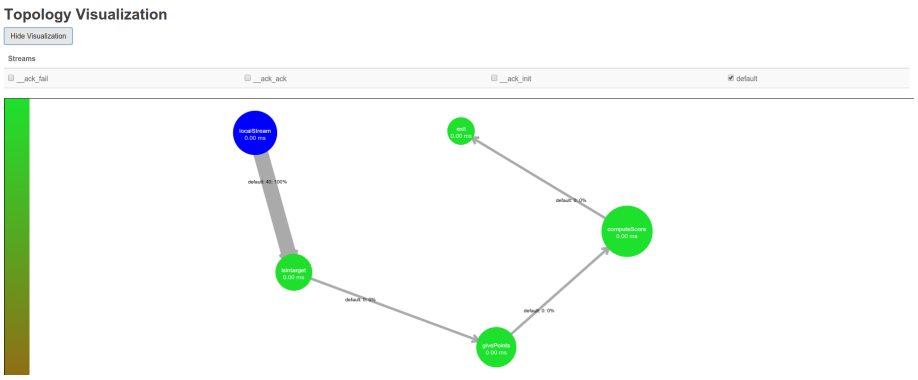


FIGURE 4 – Interface Storm UI : Visualisation d'une topologie

## 1.4 Bilan

A ce niveau du TP, vous devez pourvoir voir :

— les tuples émis par votre générateur de flux dans le terminal où vous avez exécuté le *startStream.sh*

— le trafic entre les trois opérateurs de votre topologie via l'interface ui de STORM

— les tuples émis par votre topologie dans le terminal où vous avez exécuté *main.js*

# 2 Topologies à réaliser

Remarque importante : pour ne pas gaspiller les ressources (CPU, RAM) de votre VM, et sauf contre indication, il sera important de stopper les topologies précédentes avant de lancer une nouvelle topologie (cf bouton KILL sur l'interface UI de STORM) [3].

## 2.1 Contexte

Pour cette partie du TP, nous considérons l'espace de jeu représenté sur la Figure 5. Cet espace correspond à une matrice 40 × 20 de cellules sur lesquelles va se déplacer votre pion en fonction des coordonnées qui seront produites par un flux en local sur votre VM. Le schéma de votre flux est (id, name, team, x, y, color). A noter que l'espace est en fait un tore (anneau vertical et horizontal). Par exemple un pion se trouvant en position (40,10) peut se retrouver en position (1,10) en réalisant un déplacement d'un pas à droite. A noter que la couleur du pion évolue aussi.

L'espace de jeu est composé de 5 zones spécifiques (représentées par les zones grisées sur la Figure 5 qui vous aurez à implémenter [4]), dites "zones cibles". Le principe du jeu consiste à associer des points au

3. Mettre 0 comme valeur dans le dialogue et valider

4. Avant de prendre l'option d'élargir les zones cibles pour augmenter vos chances de gagner plus de points, nous attirons votre attention sur la question 2.3.2

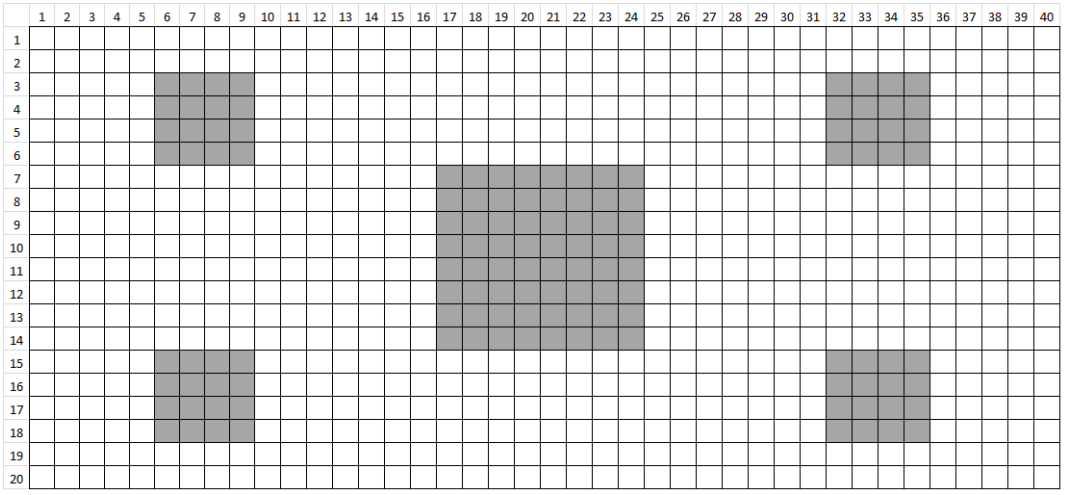

FIGURE 5 – Espace de jeu composé de 5 zones cibles.

pion quand celui-ci se trouve sur une des zones cibles. Le traitement de votre flux de coordonnées sera effectué par *Apache STORM*.

## 2.2 Traitement de topologies mono-source

### 2.2.1 Affectation des points

Il s'agit ici d'implémenter un opérateur *stateless*.
Définir un bolt, nommé "GivePointsBolt" qui détermine les points associés à un tuple. Les tuples retournés par ce bolt ont pour schéma (id, name, team, points). Si les coordonnées (x,y) correspondent à une cellule d'une zone cible, la valeur de point vaut 1 si la valeur de couleur est 'black' et 3 si cette couleur est 'red'. La valeur de point est 0 si les coordonnées (x,y) ne correspondent pas à une cellule d'une zone cible.

Définir la topologie topoT2, qui permettra de tester votre bolt "GivePointsBolt" avec *StreamSimSpout* et *ExitBolt*.

Tester votre topologie topoT2 (après avoir arrêté votre topologie topoT1).

### 2.2.2 Calcul de score

Il s'agit ici d'implémenter un opérateur *stateful*.
Définir un bolt, nommé "ComputeScoreBolt" qui calcule le score courant du joueur. Les tuples retournés par ce bolt ont pour schéma (id, name, team, score). La valeur de score correspond à la somme cumulée des points obtenus par le joueur. Un tuple sera émis à l'arrivée de chaque nouveau tuple en entrée du bolt.

Définir la topologie topoT3, qui permettra de tester votre bolt "ComputeScoreBolt" avec *StreamSimSpout* ,*GivePointsBolt* et *ExitBolt*.

Tester votre topologie topoT3 (après avoir arrêté votre topologie topoT2).

### 2.2.3 Recherche d'anomalies

Il s'agit ici d'implémenter un opérateur *stateless* avec fenêtrage.
Nous allons analyser votre flux de coordonnées afin d'identifier s'il y a ou non des anomalies. Dans votre flux, les valeurs de x (respectivement y) évoluent aléatoirement et deux tuples successifs ne peuvent avoir au plus 1 pas de différence entre leurs x (respectivement leurs y) : le pion peut reculer/remonter de 1, rester sur place ou avancer/descendre de 1. Nous définissons donc une anomalie, si un saut de plus

d'une case est effectué par un joueur.

Définir un bolt, nommé "AnomalyDetectionBolt", qui détermine si deux tuples successifs de la séquence qu'il a reçu ont bien une variation de leur x et de leur y limité à une case. Attention, il sera nécessaire de gérer les bords (Rappel : l'espace de jeu est un anneau).

Définir la topologie topoT4, qui permettra de tester votre bolt "AnomalyDetectionBolt" avec *StreamSimSpout* et *ExitBolt*.

Tester votre topologie topoT4 (après avoir arrêté votre topologie topoT3).

#### 2.2.4 Evolution du score

Il s'agit ici d'implémenter un opérateur *stateful* avec fenêtrage.
Nous allons modifier la façon de notifier le score. Définir un bolt, nommé "InfoScoreBolt" qui retourne la valeur cumulée du score (totalScore) comme vous l'avez fait dans la topologie topoT3, qui précise le score qui a été obtenu les 15 dernières secondes (lastScore) en précisant si ce dernier est croissant, décroissant ou constant par rapport à la valeur calculée lors de la fenêtre précédente (evol). Les tuples retournés auront pour schéma (idscore, name, team, totalScore, lastPoints, evol). L'identifiant (idscore) sera composé de l'identifiant du premier tuple de la fenêtre (id1) et l'identifiant du dernier tuple de la fenêtre (id2) et aura la forme "id1-id2".

Définir la topologie topoT5, qui permettra de tester votre bolt "InfoScoreBolt" avec *StreamSimSpout*, *GivePointsBolt* et *ExitBolt*.

Tester votre topologie topoT5 (après avoir arrêté votre topologie topoT4).

### 2.3 Traitement de topolgies multi-sources

Dans cette partie, nous allons analyser les flux des membres de votre groupe. Pour rappel, le groupe qui vous est attribué est défini sur la Figure 2. Il sera judicieux que l'ensemble des membres d'un groupe se synchronisent pour relancer quasi en même temps un nouveau flux, histoire que les scores comparés soient sur des échelles comparables.

#### 2.3.1 Podium du groupe

Définir une topologie topoT6 qui permet de se connecter aux flux de sortie des membres de votre groupe et d'émettre le résultat sur le port 9003 avec l'adresse IP Multicast 226.0.0.$< X >$ où $< X >$ correspond au dernier nombre de votre adresse IP. Définir un bolt, nommé "PodiumBolt", qui retourne les trois meilleurs scores calculés sur une fenêtre sautante. Les tuples retournés par ce bolt ont pour schéma (team, rank1, rank2, rank3). Modifier votre topologie topoT6 pour pourvoir exécuter l'opérateur "PodiumBolt" avec des fenêtre sautante de de durée de 20 secondes.

Tester votre topologie topoT6.

#### 2.3.2 Est-ce qu'ils/elles trichent ?

*Question Bonus* : Si vous avez des doutes sur les résultats retournés par les binômes qui sont sur le podium, définir les spouts et les bolts qui permet de vérifier si le score annoncé est possible. Pour cela, il faudra analyser le flux d'entrée et de sortie (exécution d'un équivalent de la topologie topoT3) sur les données du/des binômes suspects. Grâce à l'identifiant des tuples en sortie vous pourrez identifier les

tuples en entrée qui ont servi au calcul du score relatif (lastScore). Le schéma des tuples retournés est (name, idscore, consistent) où consistent "OK" si le score anoncé est cohérant avec les données en entrée et "Houston, We've Had a problem !" sinon.

Définir votre topologie topoT7 contrôler les scores fournis par un ou plusieurs membres de votre groupe.

## Deuxième partie

# Congestion de la topologie

L'objectif de cette partie est de vous sensibiliser aux aspects liés à l'élasticité dans STORM. Pour réaliser cette partie, vous devrez vous mettre en groupe de 2 binômes.

## 1 Contexte

Nous allons amener Storm dans ses limites afin de voir qu'une fois les stratégies d'allocations épuisées, une autre façon d'éviter la congestion d'opérateurs consiste à modifier le degré de parallélisme desdits opérateurs, et quand le changement du degré de parallélisme ne suffit plus, il est nécessaire d'allouer de nouvelles ressources. Ces nouvelles ressources, vous pourrez les récupérer grâce à l'instance de STORM se trouvant dans le répertoire "forOther" d'un des membres de votre équipe. Pour savoir quelle instance utiliser parmi les membres de votre groupe, on se base sur le principe de l'IP suivante (pour l'IP 192.168.73.X, on prend le "forOther" du membre du groupe dont l'IP est 192.168.73.Y où Y = M + ((X − M) + 1) modulo 5 avec M = $\min_{X du groupe}(X)$ la plus petite valeur de X dans le groupe. Autrement dit, pour le groupe A, le binôme dont l'IP finit par 205, utilisera le "forOther" du 206, le 206 utilisera le "forOther" du 207... et le 209 utilisera le "forOther" du 205.

## 2 Travail à réaliser

### 2.1 Topologie

Afin que les performances réseaux ne biaisent pas les tests que vous aurez à effectuer, vous utiliserez le spout *BigDataSpout* qui vous est fourni dans le package *stormTP.operator*. Les tuples émis par ce spout ont pour schéma (id, nb, value).

Nous allons dans un premier temps définir un bolt qui effectue une opération coûteuse. Définir un bolt de type 'stateless', nommé "ConsumeTimeBolt", qui pour chaque tuple reçu, prend la valeur de l'attribut "value" et permute chaque chiffre de ce nombre. Ainsi si value vaut 1234567890, la valeur retournée sera la chaine de caractère "0987654321". A chaque fois le calcul est effectuer $nb^2$ fois. Les tuples émis par ce bolt ont pour schéma (id,value, eulav).

Définir la topologie topoE1, qui permet de tester votre bolt "ConsumeTimeBolt" avec *BigDataSpout*.

En regardant dans l'ui l'affichage de votre topologie, que remarquez-vous sur l'état du noeud correspondant au bolt "ConsumeTimeBolt" (si vous ne voyez rien d'anormal... attendez un peu !).

### 2.2 Déphasage des tuples

Pour tester les performances de notre bolt "ConsumeTimeBolt", nous allons calculer le nombre de tuples déphasés (i.e. qui n'ont pas pu être traité dans le temps imparti). Pour cela, ajouter une instruction dans la méthode *fail* du spout pour pouvoir journaliser un message du type :
*"[BigDataFail] x failures after y seconds"*

avec x le numéro de l'échec et y le temps en seconde depuis le lancement de la topologie. Vous pourrez récupérer l'information via un grep récursif dans le répertoire logs/workers-artifacts/ttt où ttt correspond au nom de votre topologie.

Par la suite, nous partons du principe que la topologie est congestionné à partir de 10 échecs observés.

En combien de temps votre topologie est congestionnée ?

## 2.3 Gestion de la congestion

Pour résoudre la congestion, nous allons effectuer quelques modifications.

### 2.3.1 Modification du parallélisme

Dans un premier temps, nous allons modifier le parallélisme de l'opérateur "ConsumeTimeBolt". Modifier votre topologie topoE1, pour associer 4 workers et 4 Executor à votre opérateur.

Qu'observez-vous ?

### 2.3.2 Ajout d'un supervisor distant

Pour finir, configurer le storm du répertoire "forOther" qui vous a été attribué comme slave. Pour cela, modifier le conf/storm.yaml du slave et affecter le nom complet de votre VM (stormXXX.novalocal) au nimbus.seeds. Il est aussi nécessaire de modifier le /etc/hosts du slave pour déclarer votre VM. Il sera également nécessaire de modifier le /etc/hosts de votre VM pour déclarer le slave. Lancer un supervisor qui sera connecté à votre nimbus. Vérifier sur l'ui que le nouveau supervisor est bien répertorié dans la section "Summary supervior".

Modifier votre topologie *topoE1* pour qu'elle puissent s'exécuter sur 8 workers (et non plus 4).

Qu'observez-vous ?

=>=>=>=>=>=>F=>=>=>=>=>=>\
=>=>=>=>=>=>I=>=>=>=>=>=>=>=>=|=>=>=>=>FIN
=>=>N=>=>=>=>=>=>=>=>=>=>=>=>/