Systèmes multi agents

Décontamination de réseaux

Réalisé par
HAFDI Ahmed
EL KACEM Mouad
SADDIKI Achraf-
MEGHRICHI Imane

Encadré par
P. ELFAKER Abdel

2020-2021

Tables des figures :

Contents

1 Introduction générale

L'objectif étant de mettre en avant l'utilité des agents dans le cadre du calcul distribué et parallèle. Les agents peuvent être stationnaires ou mobiles. Il s'agit de mettre en œuvre une application sous forme d'un système multi-agents qui correspond à 10 heures de travail effectives sous la plateforme Jade.

2 Présentation de la problèmatique

Le problème de la décontamination du réseau a des applications naturelles dans la sécurité des réseaux en informatique ou en robotique. Il consiste à coordonner une équipe d'agents mobiles afin de nettoyer un réseau contaminé : • Un agent malicieux (intrus ou virus) est introduit dans le réseau. • Une équipe de bons agents (policiers) sont chargés de traquer l'« intrus » en vue de le capturer. Le réseau est modélisé par un graphe. Les agents résident sur des machines qui constituent les nœuds du graphe dont ils traversent les arêtes. L'agent intrus • Chaque sommet visité par l'intrus est contaminé. De même tout sommet adjacent à un sommet contaminé redevient lui-même contaminé à moins qu'il y ait un policier dedans. L'intrus est capturé quand il est dans un sommet qui contient un policier. L'agent Policier • Les policiers commencent tous au même sommet (réseau synchrone). Un sommet devient propre (décontaminé) dès qu'il y'a au moins un policier sur ce sommet.

Nous devons simuler un réseau de machines en un réseau de noeuds , appartenant à un arbre connexe. Ce réseau contient un virus mobile qui peut changer l'état des machines . On a aussi besoin d'un policier qui lui aussi peut être mobile sur le réseau, ce policier peut consulter l'état des machines , les nettoyer et intéragir avec les autres elements dans la machine .

On va donc réaliser notre système sur la plateforme Jape , qui implemente les spécifications FIPO .

3 Logique

Les noeuds sont des classes Java qui pointent sur des conteneurs Jade , le noeud a un paramètre estcontaminé qui defini son état de contamination
Le Main Container crée le graph et relie chaque noeud à un conteneur et crée l'agent policier fixe et le virus
L'agent policier fixe crée le policier mobile et lui passe la liste des noeuds (conteneurs) à visiter dans une logique de traverse par profondeur , son rôle consiste à netoyer les noeuds affféctées et de chercher le virus pour l'attraper (doDelete)

```
Création et initiation du policier Fixe , je vais créer le policier mobile
Création du virus et contamination du réseau
Création et initiation du policier mobile
```

Figure 1: Création et initiation des agents

Le virus traverse le graphe en largeur et contamine chaque noeud visité , la liste ordonnée de passage lui est passé en paramètre ainsi que le noeud de départ
Chaque fois que le policier mobile entre dans un conteneur il consulteà travers Des ACLMessages le autres agents résidants sur le même conteneur sur lequel il se trouve , s'il trouve un virus il le tue .

4 Les Agents :

4.1 La Classe AgentFixe

```
package agents;

import jade.content.lang.sl.SLCodec;

public class AgentFixe extends Agent {

    public void setup() {

        // register the SL0 content language and the mobility ontology
        getContentManager().registerLanguage(new SLCodec(),
                FIPANames.ContentLanguage.FIPA_SL0);
        getContentManager().registerOntology(MobilityOntology.getInstance());


        Object args[] = this.getArguments() ;
        if ( args.length == 3 ){
            try {
                AgentController mobileAgent = this.getContainerController().createNewAgent(
                        "MPolice", "agents.PoliceMobile",new Object []{ (List<Node>)args[0], args[1].toString(), (int) args[2]}) ;
                mobileAgent.start();
            } catch (StaleProxyException e) {
                e.printStackTrace();
            }
        }

    }

}
```

Figure 2:Classe AgentFixe

4.2 La Classe AgentMobile :

```java
public class AgentMobile extends Agent {


    protected List<Node> itineraire ;
    protected Node destination;


    public void foundIntrus( String loc) {
        System.out.println("Intrus found in :" + loc);
    }


    public void setup() {

        Object args[] = this.getArguments() ;
        if (args == null || args.length < 1) {
            System.out.println("Usage : <itineraire>");
            doDelete() ; //appelle takeDown()
        }
        else if ( args.length == 3) {
            System.out.println("name: " + getLocalName() );
            itineraire = (List<Node>) args[0];

            int time = (int) args[2];

            addBehaviour(new TickerBehaviour(this, time) {
                public void onTick() {

                    Iterator<Node> iterator = itineraire.listIterator();
                    try{
                        // Move to next container
                        destination =iterator.next();
                        iterator.remove();

                        //beforeMove
                        myAgent.doMove(destination.loc);
                        //afterMove

                    } catch (Exception e) {
                        myAgent.doDelete();
                    }
                }
```

Figure 3: Classe AgentMobile

5 Behaviors

5.1 GetAvailableLocationsBehaviour

Ce behavior est commun entre les agents et son rôle est de renvoyer les agents dans un conteneur.

```
public GetAvailableLocationsBehaviour(Agent a) {
    super(a, new ACLMessage(ACLMessage.REQUEST));
    request = (ACLMessage)getDataStore().get(REQUEST_KEY);

    // fills all parameters of the request ACLMessage
    request.clearAllReceiver();
    request.addReceiver(a.getAMS());
    request.setLanguage(FIPANames.ContentLanguage.FIPA_SL0);
    request.setOntology(MobilityOntology.NAME);
    request.setProtocol(FIPANames.InteractionProtocol.FIPA_REQUEST);

    // creates the content of the ACLMessage
    try {
        Action action = new Action();
        action.setActor(a.getAMS());
        QueryAgentsOnLocation q = new QueryAgentsOnLocation();
        q.setLocation(this.getAgent().here());
        action.setAction(q);
        a.getContentManager().fillContent(request, action);
    }
    catch(Exception e) { e.printStackTrace() ;}
    myAgent.send(request);
    // reset(request);
}
```

Figure 4: GetAvailableLocationsBehaviour

5.2 La classe JadeContainer

```
public class JadeContainer {


    private ContainerController container;
    public JadeContainer(String name) {
        Runtime rt = Runtime.instance() ;
        ProfileImpl profile = new ProfileImpl(false);

        //Le main container associe est deja  demarre sur localhost
        profile.setParameter(ProfileImpl.MAIN_HOST, "localhost") ;
        profile.setParameter(ProfileImpl.CONTAINER_NAME, name) ;
        container = rt.createAgentContainer(profile);
    }
    public ContainerController getContainer(){ return container; }
}
```

Figure 5: JadeContainer

5.3 La classe MainContainer

```java
public class MainContainer {


    private ContainerController mainContainer;

    public MainContainer() {
        Runtime rt = Runtime.instance();
        Properties p = new ExtendedProperties() ;
        p.setProperty("gui","true") ;
        ProfileImpl profile = new ProfileImpl(p);
        mainContainer = rt.createMainContainer(profile);
    }

    public ContainerController getContainer(){ return mainContainer; }

}
```

Figure 6: MainContainer

5.4 Le policier fixe

Le policier fixe un un seul OneShotBehavior qui le transporte vers la racine choisi du graphe.

5.5 Le policier mobile

Le policier mobile a un TickerBehaviour qui visite un noeud chaque unité de temps passée

```java
public class PoliceMobile extends AgentMobile {


    @Override
    protected void beforeMove() {
        super.beforeMove();

        if ( destination.isContaminated() ){
            System.out.println("purifying " + destination.label);
            destination.unContaminate();
        }
    }


    @Override
    protected void afterMove() {
        super.afterMove();

        getContentManager().registerOntology(MobilityOntology.getInstance());
        getContentManager().registerLanguage(new SLCodec(), FIPANames.ContentLanguage.FIPA_SL0);
        addBehaviour(new GetAvailableLocationsBehaviour((AgentMobile) this));
    }
}
```

Figure 7: Behavior de l'agent policier mobile

5.6 Le virus

Le virus a un TickerBehaviour qui visite un noeud chaque unité de temps passée et change l'etat de l'attribut isContaminated.

```java
public class VirusMobile extends AgentMobile {


    @Override
    protected void beforeMove() {
        super.beforeMove();
    }


    @Override
    protected void afterMove() {
        super.afterMove();
        destination.contaminate();

        getContentManager().registerOntology(MobilityOntology.getInstance());
        getContentManager().registerLanguage(new SLCodec(), FIPANames.ContentLanguage.FIPA_SL0);
        addBehaviour( new GetAvailableLocationsBehaviour((AgentMobile) this) );
    }


}
```

Figure 8:Behavior de l'agent virus

6 Exécution

6.1 Structure du projet

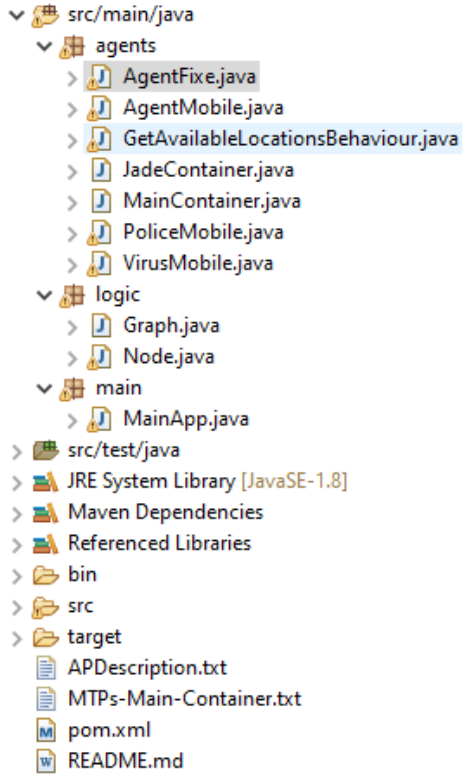

Figure 9: structure du projet

6.2 MainApp du Projet

```
public MainApp() {
    mainContainer = new MainContainer().getContainer();

    // define contaners
    containers.put("root",  new JadeContainer("root").getContainer());
    for (int i = 1; i < 7; i++) {
        containers.put("node-"+i,  new JadeContainer("node-"+i).getContainer());
    }

    // define nodes
    graph.addNode(new Node("root"));
    for( int i =1; i<6; i++ ){
        graph.addNode(new Node( "node-" + i ));
    }
    // defining edges
    graph.addEdge("root", "node-1");
    graph.addEdge("root", "node-2");
    graph.addEdge("node-2", "node-3");
    graph.addEdge("node-2", "node-4");
    graph.addEdge("node-2", "node-5");

    // define intrus
    graph.nodes.get("node-2").contaminate();


    itineraire = graph.getDFS("root");
    itineraire2 = graph.getBFS("node-2");
    try {

        fixAgent = containers.get("root").createNewAgent("FPolice", "agents.AgentFixe", new Object[]{itineraire, "police", 1000});
        fixAgent.start();

        mobileAgent2 = containers.get("node-2").createNewAgent(
                "Virus", "agents.VirusMobile",new Object []{itineraire2, "sick", 2000}) ;
        mobileAgent2.start();
    } catch (StaleProxyException e) { e.printStackTrace();
    }
}

public static void main(String[] args) {
```

Figure 10: MainAPP

6.3 Containers

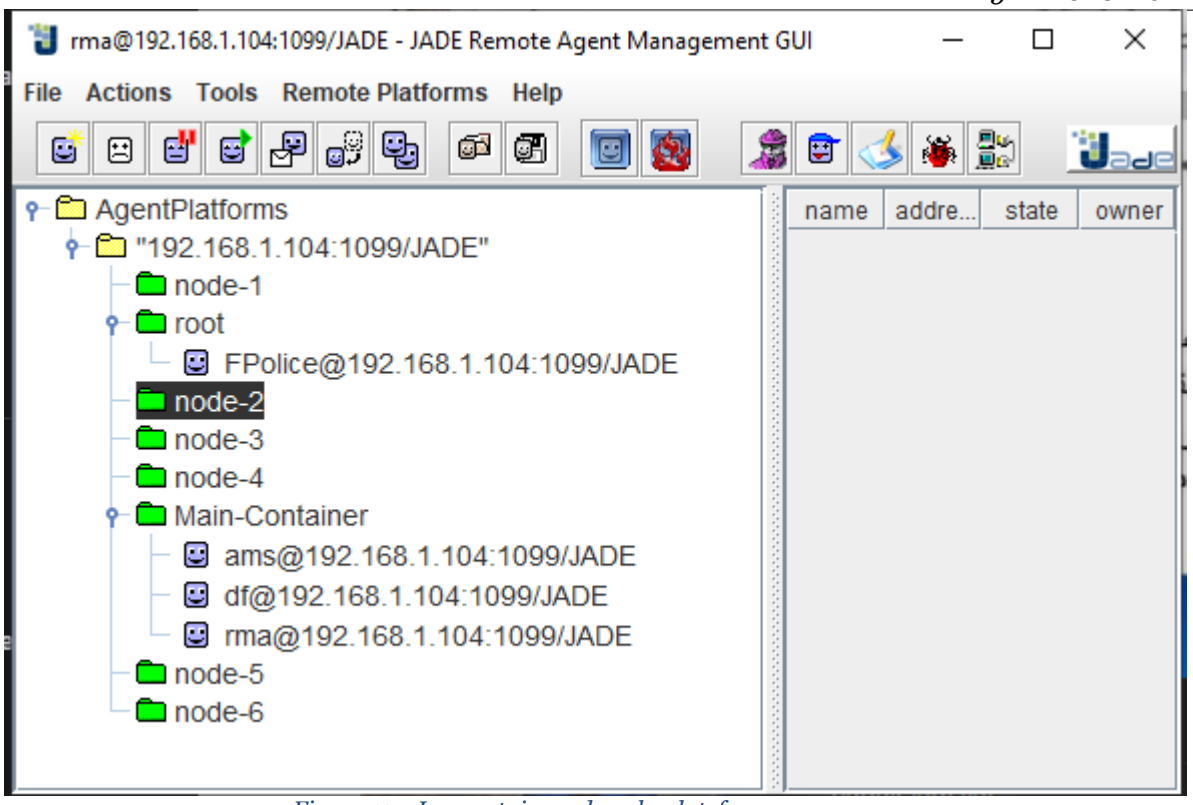

Figure 11: Les containers dans la plateforme

6.4 Scenario

Dans ce cas le virus commence au neuds 3

```
name: MPolice
name: Virus
Police ----> to : node-1
agents disponibles :
[MPolice]


purifying node-2
                                                                    Virus: ----> to : root
                                                                    agents disponibles :
                                                                   [FPolice, Virus]
                                ######################
                                -------- KILL ---------
                                -------- KILLED -------
                                ######################


Police ----> to : node-2
agents disponibles :
[MPolice]


Police ----> to : node-3
agents disponibles :
[MPolice]


                                                                    Virus: ----> to : node-3
                                                                    agents disponibles :
                                                                   [Virus, MPolice]
                                ######################
                                -------- KILL ---------
                                -------- KILLED -------
                                ######################


Police ----> to : node-4
agents disponibles :
[MPolice]
```

Figure 12: le policier attrape l'agent