Projet d'option GSI Vivaldi
Cahier d'Analyse et de Conception

Nicolas Joseph, Raphaël Gaschignard
Guillaume Blondeau, Cyprien Quilici, Jacob Tardieu

13 novembre 2013

Ce cahier d'analyse et de conception est un document qui décrit l'architecture du projet, sa décomposition en tâches ainsi que les différentes échéances et choix technologiques que nous avons fait.

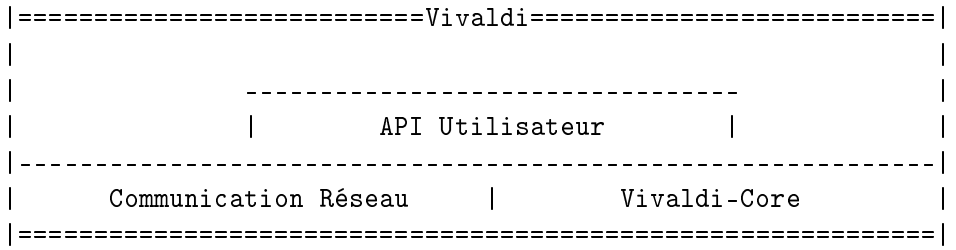
1 Dates clés

- Définition des interfaces : Samedi 16/11/2013
- Implémentation de la version alpha : 10/12/2013
- Début du travail sur le déploiement et des tests sur grid'5000 : 10/12/2013

2 Architecture Logicielle du projet

2.1 Les différentes couches logicielles

Nous avons choisi de diviser l'ensemble du projet en 2 couches logicielles réparties sur 3 briques :



2.1.1 Vivaldi Core

Cette partie contient l'algorithme vivaldi ainsi que les variables à maintenir (tableaux des plus proches voisins et RPS : Random Preloaded Subsets, ID, coordonnées dans vivaldi).

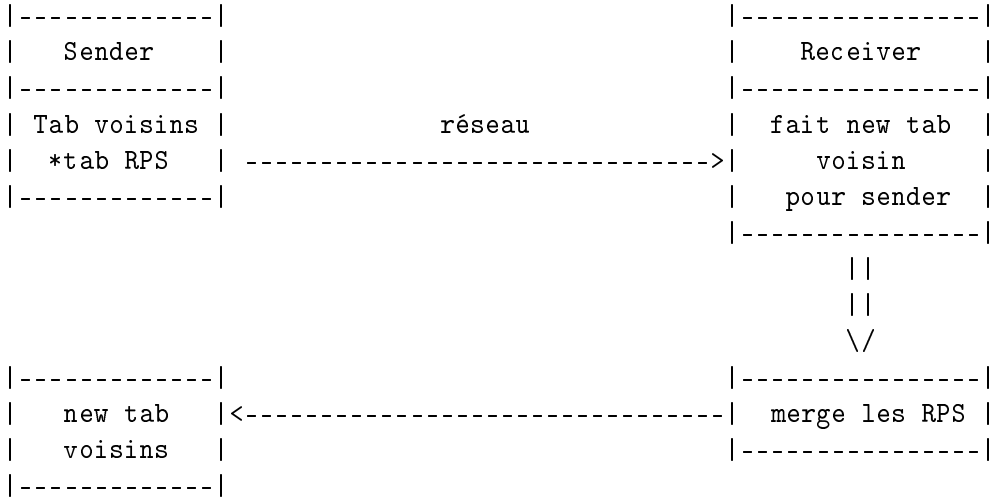
Vivaldi-core récupère n (à définir empiriquement) pings effectués par la couche réseau à chaque mise à jour du tableau de RPS et ajuste sa position dans l'espace de coordonnées vivaldi par rapport à eux.

2.1.2 Réseau

Cette partie contient la routine de maintien des RPS

Toutes les m (à définir empiriquement) secondes, la couche réseau lance une routine de mise à jour des RPS. Elle se connecte alors successivement à l (à définir) noeuds de son tableau de RPS et met à jour son tableau avec un mélange des deux tableaux du noeud courant et de son interlocuteur.

Note : à chaque fois, le noeud envoyant la requête de mise à jour du RPS envoie son ID dans le RPS de l'autre noeud pour maintenir l'information qu'il est dans le réseau. Un noeud est donc supprimé de l'union des deux tableaux.



2.1.3 API

Voici les différentes fonctions qui sont à implémenter dans l'API :

```
1 nextNode(origin: Node): Node
2 // Renvoie le noeud le plus proche du noeud origin passe en
      parametre.
3
4 nextNode(origin: Node, excluded: Set[Node]): Node
5 // Renvoie le noeud le plus proche du noeud origin passe en
      parametre en excluant la liste de noeuds excluded.
6
```

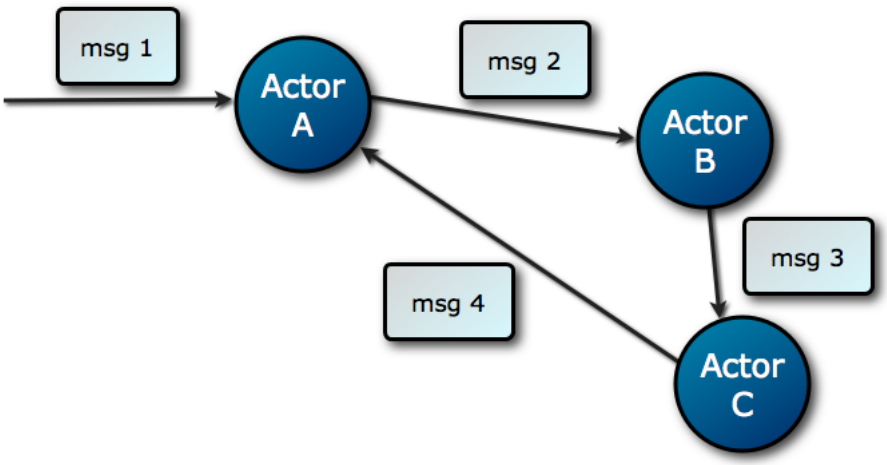
```
7 nextNodes(origin: Node, excluded: Set[Node], numberOfNodes:
      Integer): Node
8 // Renvoie les numberOfNodes noeds les plus proches du noeud origin
      en exluant la liste de noeuds excluded.
```

2.2 Le Modèle Acteur

Pour simplifier la division du travail et pour avoir un système plus modulaire, on se base sur le modèle acteur pour la coordination entre parties. Chaque système a son contexte d'exécution indépendant, et la communication entre chaque acteur se passe que par envoi de message (et pas par mémoire partagée ou verrous). Ceci nous permet de créer un programme modulaire et éviter toute une classe d'erreurs liée à la synchronisation.

Dans notre cas, on utilise la bibliothèque Akka de Scala comme infrastructure d'acteur.

FIGURE 1 – Modèle Acteur



3 Organisation de l'équipe

Le travail est divisé en plusieurs tâches. À chaque tâche est assignée un lead et un assistant. Le lead se charge de la majorité du travail de programmation alors que l'assistant aide le lead quand celui-ci en a besoin et s'occupe du "code review".

3.1 Définition des tâches

- Vivaldi-core

- Réseau
- API
- Initialisation système : écriture de la routine d'insertion dans le réseau (créations des tableaux des plus proches voisins et RPS, propagation de son identifiant, localisation dans vivaldi).
- Intégration : intégration de l'algorithme de vivaldi dans la structure logicielle existante (AkkaArc)

3.2 Répartition des tâches

	Vivaldi-core	Réseau	Initialisation Système	API	Intégration
Lead	Cyprien	Raphael	Guillaume	Nicolas	Jacob
Assistant	Nicolas	Guillaume	Jacob	Cyprien	Raphael