Simulation d'algorithmes d'équilibrage de charge dans un environnement distribué

Identifications des besoins

Kevin Barreau Guillaume Marques Corentin Salingue

3 février 2015

Résumé

Ce document dégage une première identification des besoins.

Sommaire

1 Définition du projet

1.1 Contexte

L'expansion, au cours des deux dernières décennies, des réseaux et notamment d'Internet a engendré une importante création de données, massives par leur nombre et leur taille. Stocker ces informations sur un seul point de stockage (ordinateur par exemple) n'est bien sûr plus envisageable, que ce soit pour des raisons techniques ou pour des raisons de sûreté (pannes potentielles par exemple). Pour cela des systèmes de stockages dits distribués sont utilisés en pratique afin des les répartir sur différentes unités de stockages.

Définition Un environnement distribué est constitué de plusieurs machines (ordinateurs), appelées *noeuds*, sur lesquelles sont stockées des données.

Définition Une donnée est une suite binaire de 0 et de 1 dont le contenu n'est pas important pour l'application.

Le client souhaite répartir toutes ces données de manière équitable entre les noeuds. De plus, ces données doivent être accessibles afin de pouvoir les requêter et récupérer de l'information.

Définition Une requête est un message envoyé à une machine (ou plusieurs machines) afin de récupérer ou de modifier de l'information sur des données. Nous noterons que la nature de l'information est inutile pour le bon fonctionnement de l'application.

Pour répartir toutes ces données, notre client a développé de nouveaux algorithmes d'équilibrage de charge et de réplication qu'il souhaite tester dans un environnement distribué.

Définition Une *charge* est associée à un noeud et désigne le nombre de requêtes que le noeud doit traiter.

Définition La réplication d'une donnée consiste à faire des copies de cette donnée sur d'autres noeuds.

1.2 Finalité

Nous devons développer une solution logicielle permettant de tester ces nouveaux algorithmes d'équilibrage de charge et de réplication dans un environnement distribué.

Définition Un réseau est un ensemble de noeuds qui sont reliés entre eux (en général par Internet) et qui communiquent ensemble afin de traiter toutes les requêtes reçues.

Définition La topologie d'un réseau représente l'architecture physique ou logicielle des liens entre les noeuds. Elle comporte aussi des informations sur la hiérarchie des noeuds, le placement spatial et les divers équipements reliant les noeuds.

Définition L'état d'un réseau est l'ensemble des informations caractérisant un réseau (topologie par exemple) ainsi que des informations sur les noeuds (comme leur charge actuelle).

Définition Un jeu de données est un ensemble de données dont on connait la position sur les noeuds (la donnée numéro X est sur le noeud numéro Y), éventuellement, leur contenu et qu'on est capable d'exporter et de reproduire. L'export pourra se faire sous la forme d'un jeu de requêtes pour placer les données choisies.

Définition Un jeu de requêtes est un ensemble de requêtes qu'on est capable d'exporter sous forme d'un programme ou d'un fichier qui sera exécuté par la solution et qui peut être reproduit.

Cette solution doit permettre le paramétrage d'un réseau, c'est-à-dire le nombre de noeuds souhaité et la topologie du réseau. Le client pourra simuler différents jeux de données et jeux de requêtes sur ce réseau. Cela permettra de comparer l'efficacité de ces algorithmes avec le même environnement (même jeux de données et de requêtes). Il pourra tester ses algorithmes implémentés. Enfin, il pourra visualiser la topologie et l'état du réseau *à tout moment*.

2 Hiérarchisation des besoins

Nous avons dégagé des précédentes réunions, une liste de besoins fonctionnels et non-fonctionnels. Pour mieux les comparer, nous les avons hiérarchisés en fonction de leur priorité et de leur criticité.

2.1 Priorité

La priorité est un indicateur de l'ordre dans lequel nous devrons implémenter les besoins afin de satisfaire au mieux les exigences du client.

Valeur	Signification	Description
1	Priorité haute	A implémenter dans les premiers temps
2	Priorité moyenne	A implémenter
3	Priorité faible	A implémenter (en fonction du temps restant)

2.2 Criticité

Le niveau de criticité d'un besoin est un indicateur de l'impact qu'aura la non-implémentation de ce besoin sur le bon fonctionnement de l'application.

Valeur	Signification	Description
1	Criticité extrême	L'application ne fonctionnera pas
2	Criticité haute	Certaines fonctionnalités de l'application ne fonctionneront pas
3	Criticité moyenne	Certaines fonctionnalités seront perturbées
4	Criticité faible	L'application fonctionnera correctement

3 Besoins fonctionnels

3.1 Environnement distribué

Nous évoluerons dans un environnement distribué (*Priorité* : 1, *criticité* : 1) constitué de n noeuds de stockage dans lequel on souhaite stocker m objets. En effet, l'application doit permettre de tester des algorithmes d'équilibrage de charges et de gestion de copies qui ne peuvent fonctionner uniquement dans un environnement distribué.

3.2 Gestion d'un réseau

Comme défini précédemment, un réseau est un ensemble de noeuds qui sont reliés entre eux (en général par Internet) et qui communiquent ensemble afin de traiter toutes les requêtes reçues.

3.2.1 Gestion des noeuds

Un noeud est une machine (ordinateur généralement) pouvant stocker des données et traiter des requêtes. Un noeud possède des données locales propres au fonctionnement du noeud (comme la charge par exemple).

Création d'un noeud La création d'un noeud (*Priorité* : 1, *criticité* : 1) se fait au moment où le réseau n'existe pas encore. Le projet ne prend en compte ni la création dynamique de noeud (création et ajout d'un noeud après que le réseau soit créé), ni la suppression de noeud. Il est possible de séparer ce besoin en plusieurs sous-besoins :
— Créer un noeud dans l'environnement de simulation
— Initialiser les données locales d'un noeud

Données locales d'un noeud Chaque noeud doit contenir les données locales (*Priorité* : 1, *criticité* : 1) suivantes :
— 1 vecteur correspondant à la charge de tous les noeuds
— 2 vecteurs correspondant à la popularité de chaque objet
— 1 file d'attente de message
— la requête en cours de traitement

Mise à jour des données Afin de connaître l'état du réseau de manière précise, les données locales doivent être mise à jour à chaque action (*Priorité* : 1, *criticité* : 1) . Une mise à jour à donc lieu à l'arrivée d'un message dans la file d'attente

Récupération de l'état du réseau L'application doit permettre la description de l'état du réseau (*Priorité* : 1, *criticité* : 1) . On souhaite connaître :
— la charge des noeuds
— le nombre de requêtes en attente
— la popularité des objets
Une partie de ces données font ensuite l'objet d'un affichage. [1]

3.2.2 Gestion des objets

Un objet est...

1. Objet de la partie 3.5 Visualisation des données

Une fois l'ensemble des noeuds crée, on doit pouvoir affecter à chaque noeud des objets.

Création d'un objet A détailler

Suppression d'un objet A détailler

3.2.3 Popularité d'un objet

Les algorithmes du client nécessitent de connaître la popularité d'un objet dans le réseau (*Priorité* : 1, *criticité* : 1) . La popularité d'un objet est fonction du nombre de requêtes sur cet objet. Plus ce nombre de requêtes est grand, plus l'objet est populaire.

Algorithme La calcul de la popularité nécessite l'implémentation de l'algorithme d'approximation Space-Saving Algorithm [ADA05].

3.2.4 Topologie du réseau

Un noeud maître est un noeud connu de tous les autres noeuds dans le réseau. Un noeud communique à un intervalle de temps régulier avec les autres noeuds du réseau. Tous les noeuds du réseau possèdent des informations sur les autres noeuds du réseau.

Communication Les données locales d'un noeud doivent être communiqué à un noeud maître (*Priorité* : 1, *criticité* : 1) . Le noeud maître peut centraliser toutes les données locales de chaque noeud et ainsi connaître l'état du réseau.

Paramétrage Lorsqu'il n'y aucune requête à traiter sur le réseau, l'état du réseau est stable. Plus précisément, les données locales de chaque noeud ne changent pas. Malheureusement, ils continuent de communiquer entre eux. Pour éviter ces échanges de messages inutiles, on souhaite pouvoir paramétrer l'intervalle de temps de communication (*Priorité* : 2, *criticité* : 4) .

Stockage Stocker les informations des autres noeuds (*Priorité* : 1, *criticité* : 2) (???)

3.2.5 Sauvegarde d'un réseau

Objet de la prochaine réunion

3.2.6 Importation d'un réseau

Objet de la prochaine réunion

3.3 Algorithmes

3.3.1 Algorithmes à implémenter

Définitions La charge minimum est ...

La charge moyenne est ...

Les algorithmes d'équilibrage de charge à implémenter sont (extrait du modèle du client) :

— **SLVO** (*Priorité* : 1, *criticité* : 1)
  — Déterminer la charge minimum courante.
  — Si la charge du noeud est inférieure ou égale à la charge minimum, il s'affecte toutes les requêtes en attente et en avertit les autres noeuds.

— **AverageDegree** (*Priorité* : 1, *criticité* : 1)
  — Déterminer la charge moyenne courante
  — Si la charge du noeud est inférieure ou égale à la charge moyenne, il s'affecte toutes les requêtes en attente et en avertit les autres noeuds.

Gestion des copies Entité à part entière ? (prochaine réunion)

3.3.2 Conformité des implémentations

La conformité des algorithmes implémentés est assurée par des jeux de tests suivant la démarche :

— Créer un jeu de test à la main, comprenant les paramètres de création et l'ensemble des requêtes à traiter
— Faire fonctionner l'algorithme à la main
— Stocker l'état final du fonctionnement de l'algorithme
— Faire valider le processus à la main par le client
— Lancer la simulation de l'algorithme avec le jeu de test créé à la main
— Vérifier les résultats constatés avec les résultats attendus

Si il y a une différence entre le résultat de l'éxécution de l'algorithme à la main et le résultat de l'éxécution par simulation, une vérification par le client peut être envisagée dans le cas de résultats *presque* similaires. La notion de similitude est laissée à l'appréciation de l'équipe en charge du projet, lors de la vérification.

3.3.3 Autres algorithmes

L'application doit permettre aussi avoir la possibilité de tester d'autres algorithmes. Notre solution doit donc proposer un système permettant d'implémenter ces algorithmes et de les tester. (*Priorité* : 3, *criticité* : 3)

3.4 Simulation de requêtes

Une requête est un message envoyé à une machine (ou plusieurs machines) afin de récupérer ou de modifier de l'information sur des données.

3.4.1 Création d'une requête

Une fois le réseau crée, l'utilisateur peut en

3.4.2 Sauvegarde d'un jeu de requête

Objet de la prochaine réunion (SENS ?)

3.4.3 Importation d'un jeu de requête

Objet de la prochaine réunion (SENS ?)

3.5 Visualisation des données

— Temps de réponse moyen sur les requêtes passées.
— Charge d'un noeud
— Popularité des objets
Note Bien définir ces items.

4 Besoins non fonctionnels

4.1 Cassandra

Cassandra est une base de données distribuée. Nous créons notre environnement de simulation à partir de la dernière version stable, `Cassandra 2.1.2`.

Le choix de cette solution nous a été fortement recommandé par le client. En effet, celui-ci dispose de connaissances sur cette application et pourra donc plus facilement intervenir s'il souhaite faire évoluer le projet en implémentant par exemple de nouveaux algorithmes.

4.2 Gestion d'un réseau

4.2.1 Communication entre noeuds

Pour connaître l'état du réseau, il faut regrouper les données locales des noeuds. Nous cherchons donc a récupérer ces données en un temps raisonnable ($O(log(n))$ pour n noeuds).

Pour cela, nous nous appuierons sur le protocole `Gossip` [Fou14a]. Périodiquement, chaque noeud choisi n noeuds aléatoirement dont un noeud *seed*, noeud en mesure d'avoir une connaissance globale du système [Fou14b], et il communique à ces noeuds ses statistiques (valeur de sa charge, objets les plus populaires...).

Ainsi, la connaissance globale du système se fait en $O(log(n))$.

4.2.2 Taille des données

Objet prochaine réunion

4.3 Visualisation des données

Une vue correspond à une fenêtre de l'application, c'est à dire ce que voit l'utilisateur.

4.3.1 Etat du réseau

La vue permet de montrer l'état du réseau.

Le réseau est représenté par un graphe, les machines par des noeuds. Pour chaque machine, les données affichées sont la charge ainsi que le contenu de la file d'attente.

4.3.2 Actualisation de la vue

L'état du réseau doit-être visible en temps réel.

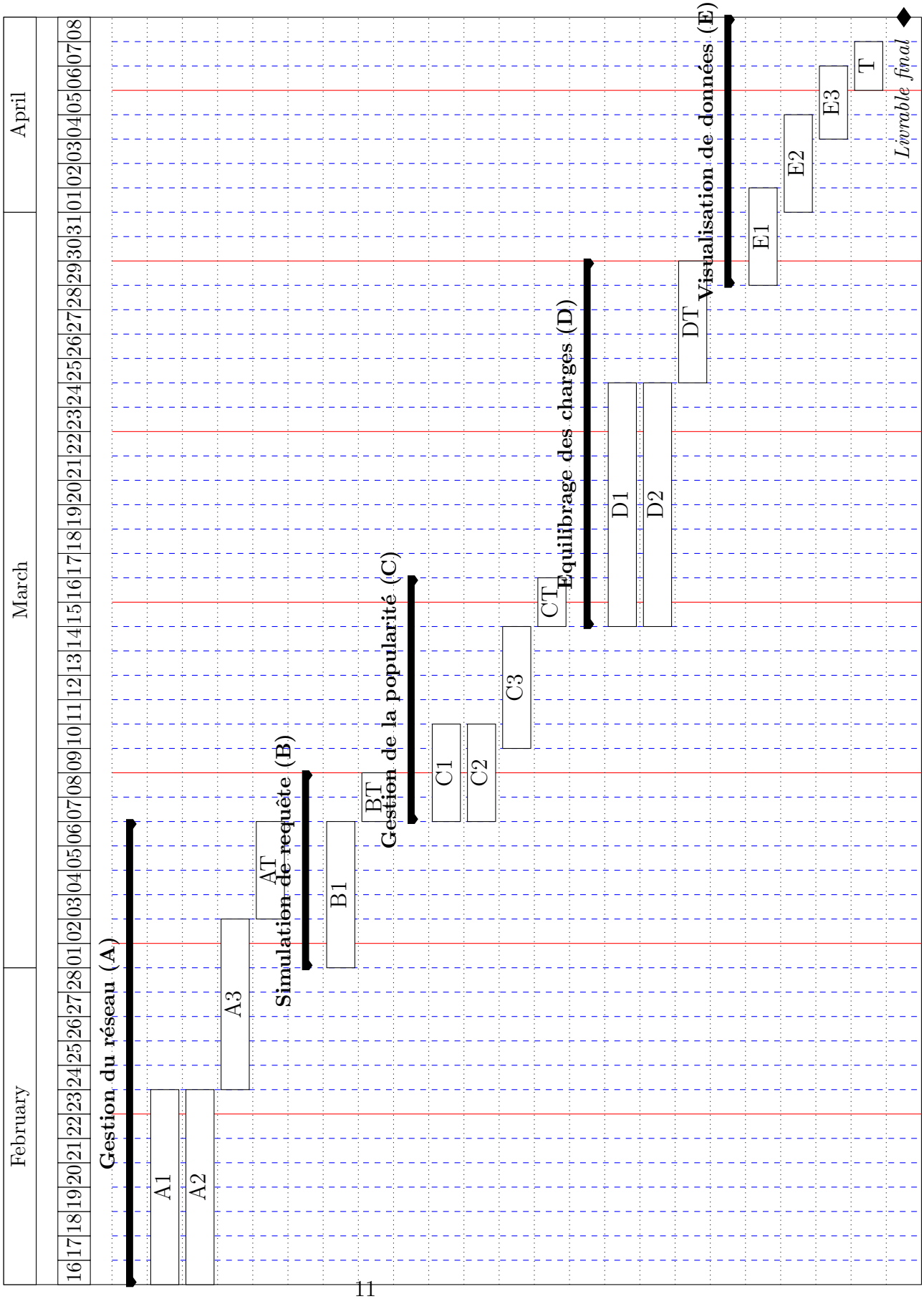
La vue peut donc être actualisée toutes les 0.5 secondes. Un délai plus faible risquerai de la rendre invisible (données clignotantes sur l'écran).

4.4 Maintenabilité du code

Nous ne pensons pas que le projet sera totalement terminé le 8 Avril 2015, date de rendu du code et du mémoire. Pour cela, nous avons défini quelques normes pour que le projet puisse être repris. (a détailler)

5 Répartitions des tâches

5.1 Diagramme de Gantt



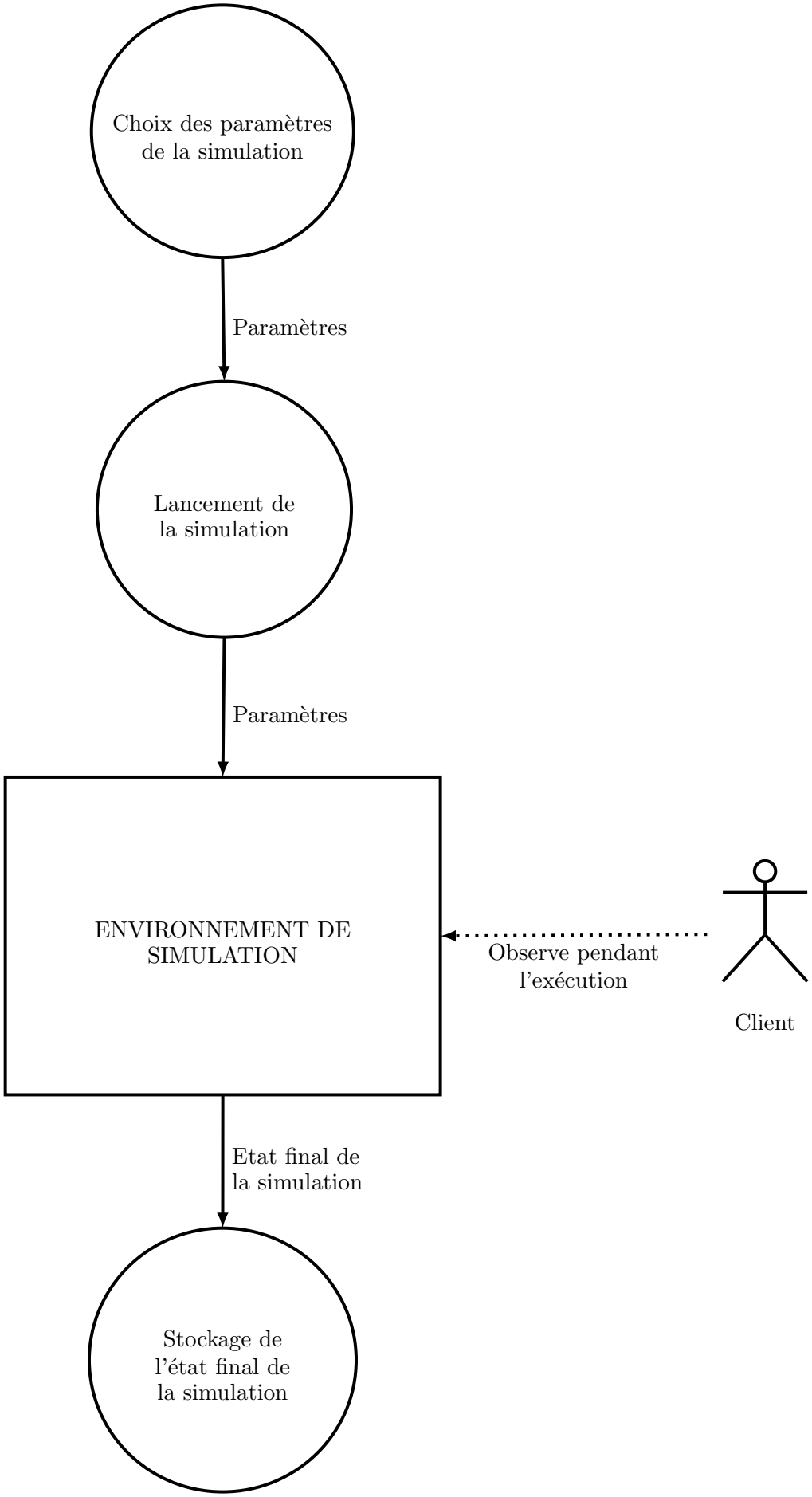
5.2 Affectation des tâches

Fct	Description	Développeur(s)	Commentaire
A1	Création des noeuds		
A2	Données locales des noeuds		Initialisation et implémentation
A3	Communication entre noeuds		
AT	Tests groupe A		Vérification, tests, mémoire
B1	Simulateur jeu de requêtes		A détailler
BT	Tests groupe B		Vérification, tests, mémoire
C1	Popularité objet sur noeud		
C2	Space-Saving Algorithm		
C3	Popularité d'un objet		
CT	Tests groupe C		Vérification, tests, mémoire
D1	Implémentation SLVO		
D2	Implémentation AverageDegree		
DT	Tests groupe D		Avec client
E1	Prise en main Tulip		
E2	Représentation réseau		
E3	Représentation données		
T	Tests finaux		Vérification, tests, mémoire

6 Livrables

6.1 Livrable “final”

Il devra être remis le 8 Avril 2015.


Choix des paramètres de la simulation
Paramètres
Lancement de la simulation
Paramètres
ENVIRONNEMENT DE SIMULATION
Observe pendant l'exécution
Client
Etat final de la simulation
Stockage de l'état final de la simulation

Références

[ADA05] Metwally A, Agrawal D, and El Abbadi A. Efficient computation of frequent and top-k elements in data streams. 2005.

[Fou14a] The Apache Software Foundation. Architecturegossip - cassandra wiki. <http://wiki.apache.org/cassandra/ArchitectureGossip>, 2014. [Accessed 21 January 2015].

[Fou14b] The Apache Software Foundation. Faq cassandra wiki. <http://wiki.apache.org/cassandra/FAQ#seed>, 2014. [Accessed 21 January 2015].