Rapport et réponses aux questions
TP2 : Services distribués et gestion des pannes
INF4410 : Systèmes répartis et infonuagique

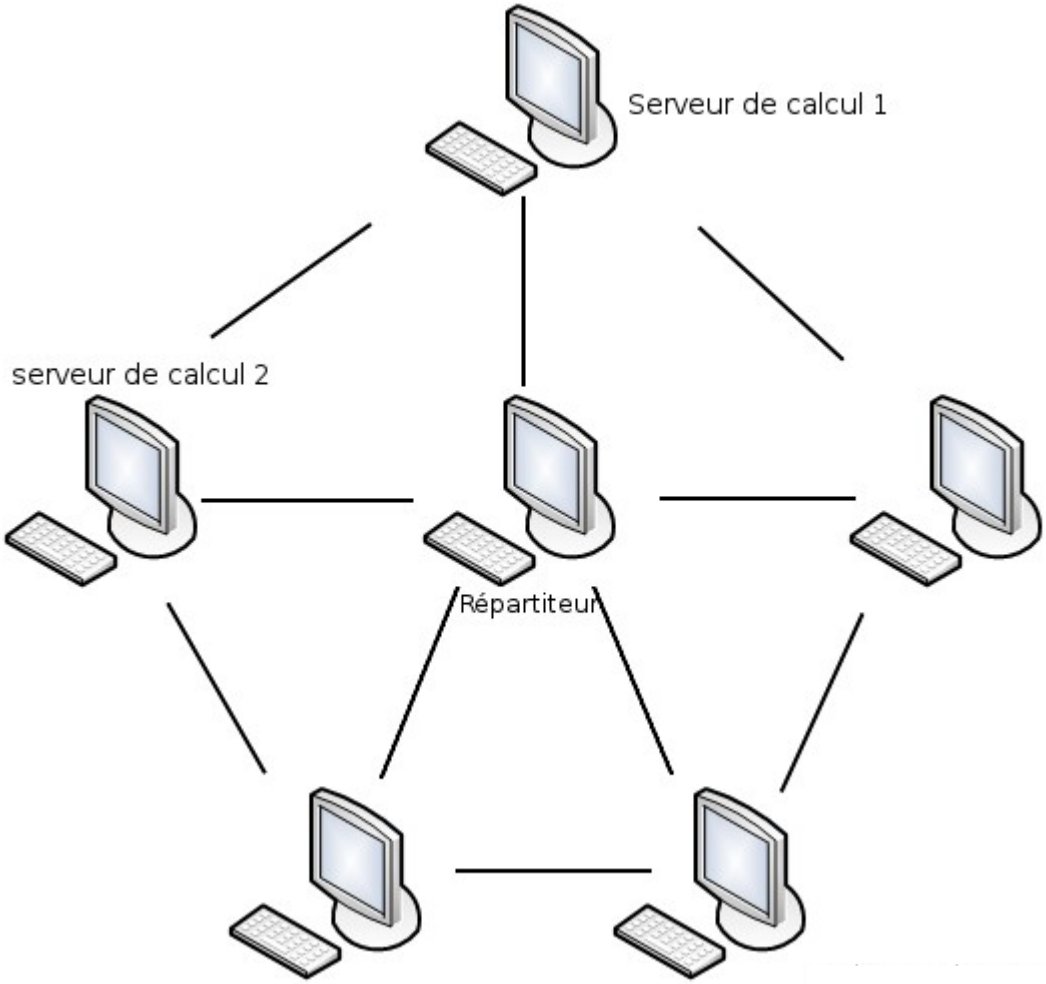
6/11/2013

Langrenay Pierre 1699852
Lottier Jean - Michel

Langage utilisés : Java & socket

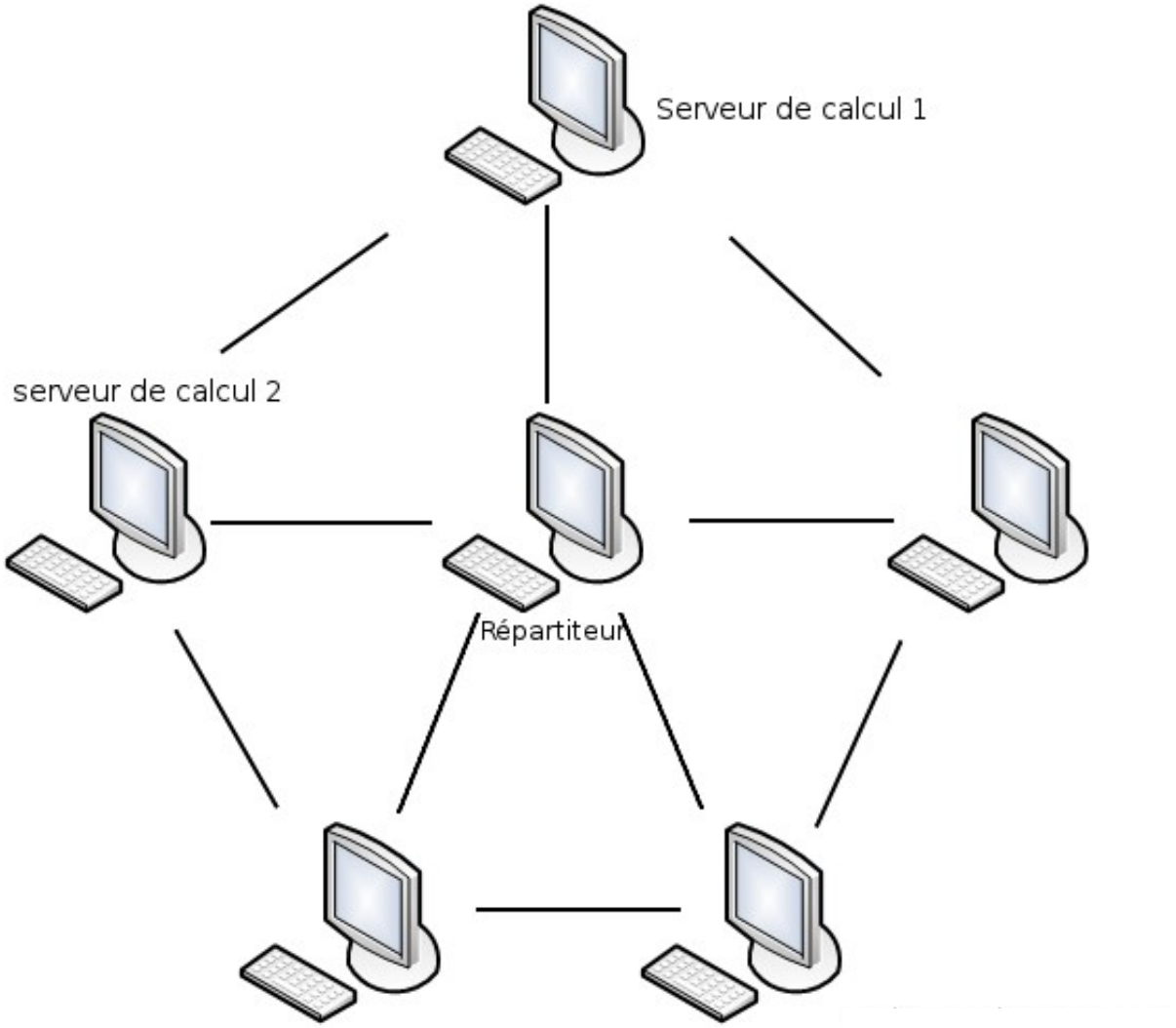
Question :

L'architecture qui nous a été demandé d'implémenter est un réseau en étoile, voir ci dessous :



Ce type de réseaux présente l'avantage qu'il est facile de rajouter des postes. Il est également possible d'identifier une panne qui proviendrait d'un serveur de calcul, par ailleurs si une telle panne se produisait, le réseau ne serait pas paralysé.

Notre solution serait la suivante :



De cette façon, en cas de panne du répartiteur, les serveurs de calcul peuvent toujours communiquer entre eux et élire un nouveau répartiteur.
On pourrait imaginer qu'en cas d'engorgement du répartiteur, un serveur de calcul qui serait incapable de traiter sa tâche pourrait communiquer cette dernière à un serveur de calcul voisin.
Un inconvénient serai que l'on complexifie notre système. Chaque serveur de calcul prend de l'importance et acquiert une certaine autonomie. Les scénarios qui provoqueraient une panne du système sont une déconnexion partielle du réseau, provoquant deux réseaux différents, incapable de discuter entre eux.

Rapport :

Toutes les requêtes qui transitent entre le dispatcher et les serveurs de calcul sont stockées dans un journal, tous les serveurs ont accès au journal. Il existe différents types de commandes : init, task, abort, commit, close, fail.

Init : initialisation de la connexion entre le dispatcher et un serveur de calcul
Task : le dispatcher communique au serveur de calcul, le travail qu'il doit réaliser
commit : le serveur de calcul renvoie le travail qu'il a effectué, tout ces biens passé
Abort : le serveur de calcul n'a pas réussi à calculer sa portion de texte car il ne dispose pas de suffisamment de ressources.
Fail : le serveur de calcul n'est plus opérationnel (crash éventuel).
Close : clôture de la communication entre le dispatcher et le serveur de calcul

Arborescence du journal :

• Journal

pour éviter tout conflit de corruption de données, nous avons utilisé un sémaphore, plus particulièrement un mutex (une seule ressources).

le journal contient les informations suivantes :
- l'ensemble des tâches qui ont été envoyées par le dispatcher ou les serveur de calcul
private ArrayList<String> tasksTotal;

- les informations sur chacun des serveurs de calcul
private HashMap<Long, ServerInfo> serverInfos;

long : permet de numéroter les serveurs

• ServerInfo

ServerInfo contient des informations relatives à chacun des serveurs
- ID
- Nom
- Adresse IP
- Port
- Quantité de ressources qu'il possède
- toutes les tâches qu'il doit effectué
- L'ensemble des requêtes que le dispatcher a envoyé au serveur de calcul
il s'agit du map, les requêtes sont triées par type : init, task, abort, commit, close, fail
private HashMap<Commands.commands, ArrayList<Request>> requests;

- L'ensemble des requêtes que le dispatcher a recu du serveur de calcul, toujours trier par type (voir ci-dessus)
private HashMap<Commands.commands, ArrayList<Request>> responses;

• Request

- un entier TaskID
- un entier indiquant la quantité de ressources nécessaire pour traiter la tâche.
- le type de tâche : init, task, abort, commit, close, fail

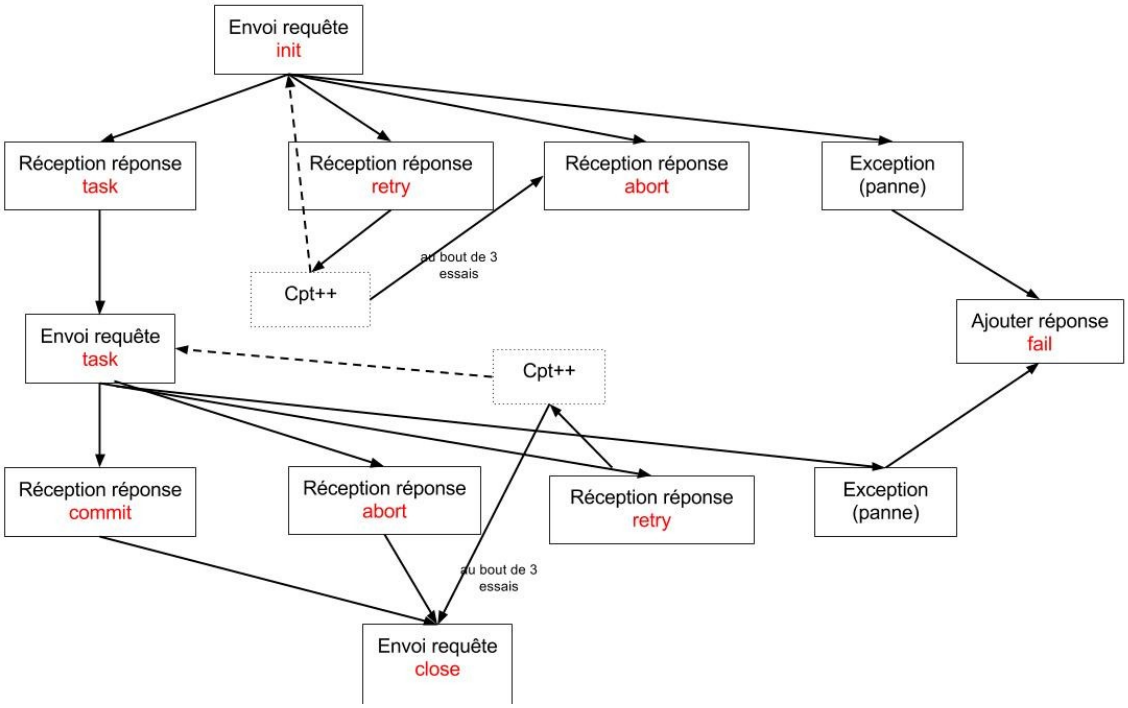
- le résultat de la requête stocké sous forme de HashMap
private HashMap<String, Integer> result;

String : un mot du texte
Integer : nombre d'itération du mot dans le texte étudié

Pour communiquer les portions de textes que chaque serveurs de calcul doit effectuer, nous avons décidé de communiquer le texte entier à chacun des serveurs. Pour préciser les portions de texte que chaque serveur doit étudier, nous indiquons une ligne de début et une ligne fin. Ces deux valeurs apparaissent dans l'objet Request :

private int firstLineNumber;
private int lastLineNumber;

Le graphique ci- dessous identifie les différentes contraintes et leurs solutions dans la communication entre les serveurs.



Test de performance – mode sécurisé

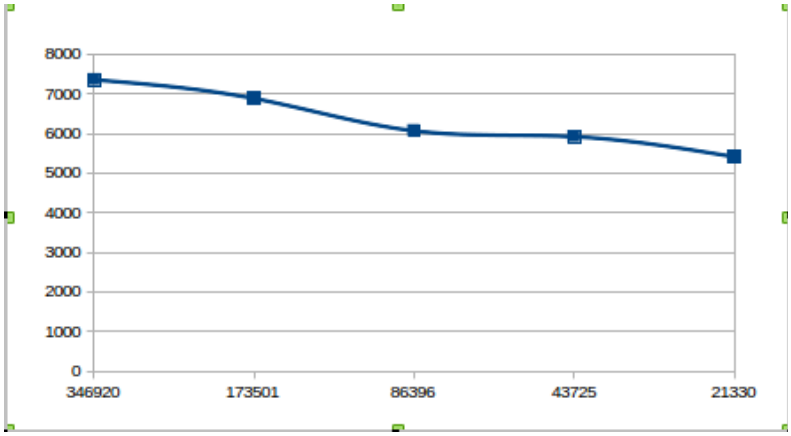


Illustration 1: Q1=25000

en abscisse, la taille en octet du texte a étudié, en ordonné le temps d'exécution.

On remarque que lorsque les serveurs de calcul ont peu de ressources, le temps d'exécution devient plus court. Cela revient à dire qu'effectuer beaucoup de tâches élémentaires est plus rapide qu'effectuer quelques macro-tâches.

Test de performance – mode non sécurisé

Nous nous sommes penché sur le mode non sécurisé, cependant nous avons eu des erreurs que nous n'avons su solutionner. Par conséquent nous avons été dans l'impossibilité de réaliser le graphique du temps d'exécution en mode non sécurisé.