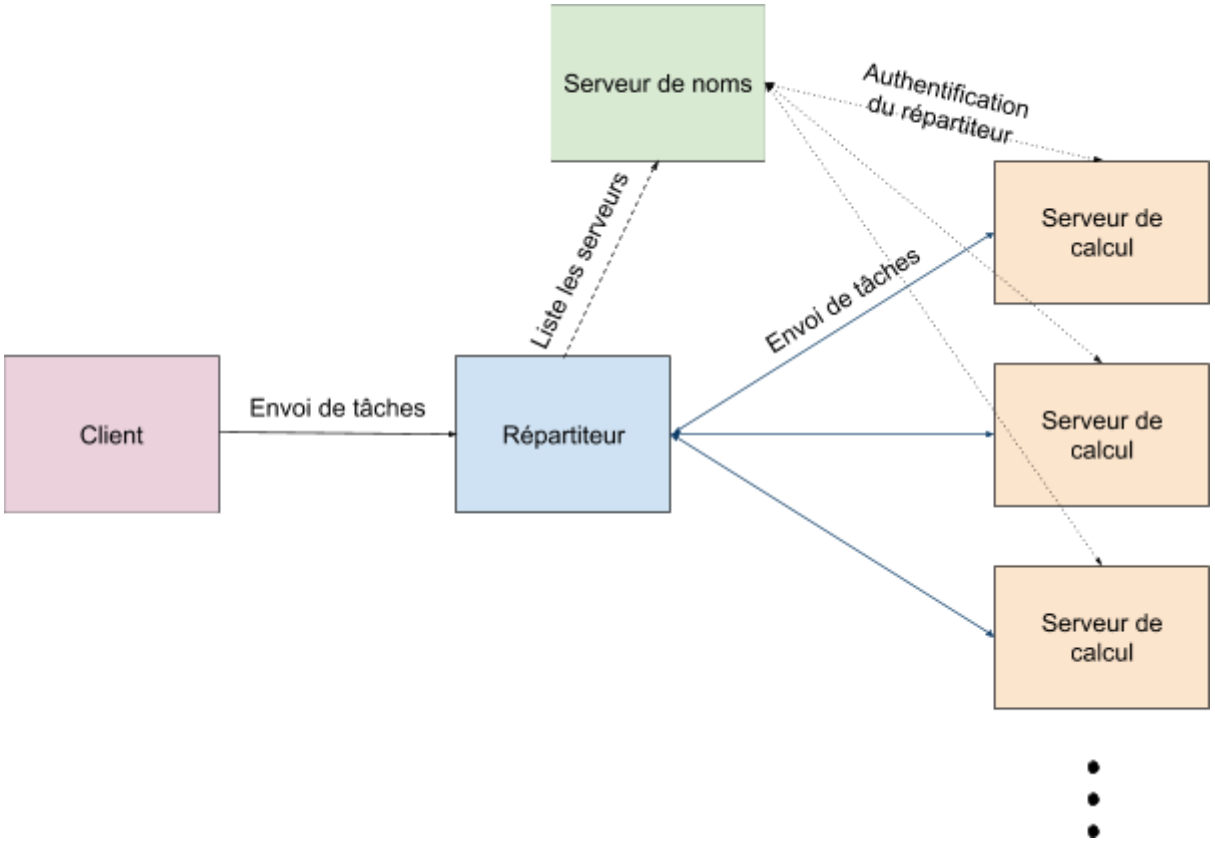
INF8480 - Systèmes répartis et infonuagique

TP2 -Services distribués et gestion des pannes

Philippe Carphin - 1693091
Alexandre Jouy - 1858162

Question 1

Notre architecture est actuellement composée d'un répartiteur, d'un serveur de nom et de multiples serveurs de calcul.

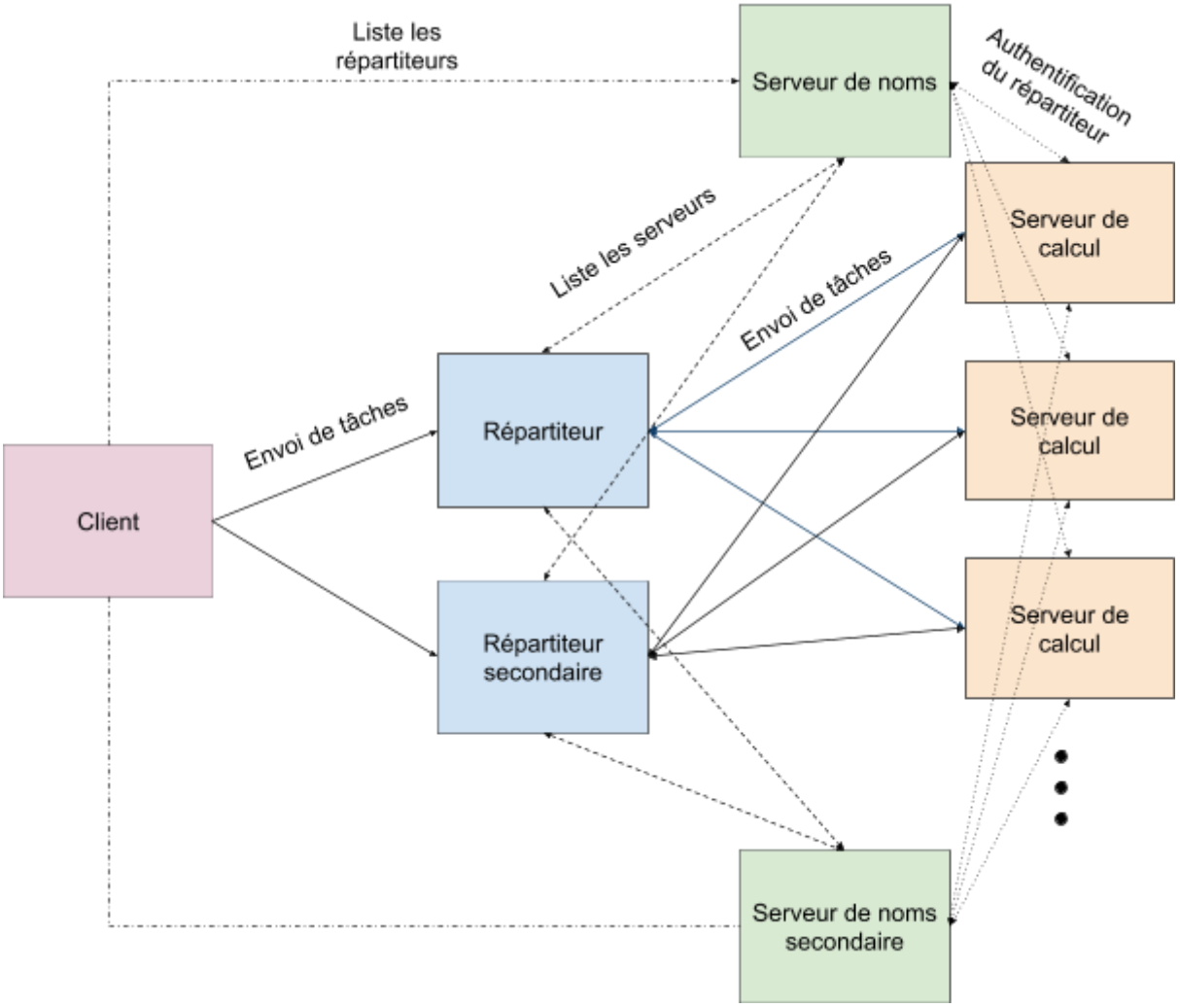


Ainsi, le client envoie une liste de calculs à effectuer au répartiteur qui à son tour répartit les calculs à faire entre les différents serveurs disponibles. La liste de ces serveurs est disponible auprès du serveur de nom et peut être récupérée par le répartiteur.

Dans une architecture répartie, le ou les points faibles se situent au niveau des éléments uniques de l'installation. Ainsi dans notre cas on identifie deux points faibles principaux : le répartiteur et le serveur de noms. En effet, si l'un des deux serveurs vient à tomber en panne, il n'est alors plus possible au client de faire ses calculs.

Pour éviter cela, une solution simple à mettre en place serait de créer une redondance de ces deux serveurs. C'est à dire d'avoir au moins une instance supplémentaire à la fois du serveur de noms et du répartiteur.

On aurait alors une architecture de ce type :



On a ainsi à la fois une redondance du répartiteur et du serveur de noms. De plus l'ajout d'une fonction aux serveurs de noms permettant de lister les répartiteurs pourrait permettre au client d'obtenir l'adresse du répartiteur secondaire si le premier vient à lâcher. Cette redondance permet ainsi de pouvoir continuer à effectuer des calculs même si l'un des serveurs vient à lâcher. De plus une telle architecture permettrait également de faire de la répartition de charge si nécessaire.

Rapport

Dans ce rapport nous allons expliquer les méthodes à distance utilisées ainsi que les choix faits concernant le design des éléments critiques tels que le dispatcher.

Client

Le client est composé de la classe Client.java et son usage est le suivant :

./client [ip du répartiteur] [fichier d'opérations à lire]

Le client est un des éléments les plus simples de ce système réparti. En effet, son rôle consiste à envoyer au répartiteur les opérations dont il veut obtenir les résultats. Pour cela il lit le fichier en paramètre puis fait un appel à distance de la méthode **dispatchTasks** du répartiteur. Une fois les calculs effectués par les serveurs, le client affiche alors tous les résultats reçus.

Répartiteur

Le répartiteur est composé des classes Dispatcher.java, ComputeCallable.java et implémente l'interface DispatcherInterface.java. Son usage est le suivant :

./server [hostname] [ldap-hostname] [utilisateur] [mot de passe]

Cette classe s'occupe de faire la répartition des tâches que lui a envoyés le client. Pour cela elle implémente la méthode **dispatchTasks** de l'interface DispatcherInterface.java. De plus la classe ComputeCallable.java implémente l'interface Callable et contient donc la méthode **call**.

call permet à la méthode à distance **compute** d'être appelée sur un autre fil d'exécution après avoir instancié un *ExecutorService*. De cette manière plusieurs tâches de calculs peuvent être lancées en parallèle plutôt que devoir attendre la fin de l'exécution de la précédente.

Le répartiteur, s'il reçoit une requête pour du traitement sécurisé, consulte le service de noms pour obtenir la liste des serveurs disponibles. Ensuite, il consulte les serveurs pour leur demander leur capacité de calcul. Il divise alors les opérations entre chaque serveur et lance des threads qui s'occuperont de passer le travail à chaque serveur par petites portions selon leur capacités respectives. Il attend que chaque thread ait fini et rassemble les résultats.

Dans le calcul non-sécurisé, le répartiteur brise le travail en beaucoup de sections de taille égale à la capacité minimale des serveurs. Pour chaque morceau, il fait alors faire le travail

à tous les serveurs. Ensuite il compare les réponses et par démocratie, rejette les réponses “malicieuses”.

Serveur de noms

Le serveur de nom est composé de la classe LDAP.java et implémente l’interface LDAPInterface.java. Son usage est le suivant :

```
./LDAP
```

Cette classe contient la liste des serveurs de calculs ainsi que les couples utilisateur/mot de passe pouvant être demandés lors d’un calcul en mode sécurisé. Elle implémente également les méthodes **authenticate** et **listServers** de l’interface LDAPInterface.java.

authenticate prend comme arguments un utilisateur et un mot de passe et les compare à ceux stockés dans la classe. Si l’authentification est réussie elle renvoie alors *true*. Cette méthode permet d’authentifier le répartiteur lors d’un calcul.

listServers permet de retourner la liste des serveurs stockée dans la classe. Cette méthode peut être appelée par le répartiteur.

Serveur de calculs

Le serveur de calcul est composé des classes Server.java, qui définit ses fonctionnalités, Operations.java qui définit les opérations mathématiques qu’il effectue et implémente l’interface ServerInterface.java. Son usage est le suivant :

```
./server [taux d’erreur] [port]
```

ServerInterface.java implémente la méthode **compute** qui prend en paramètre les opérations à effectuer, le mode de fonctionnement ainsi qu’un nom d’utilisateur et un mot de passe. Cette méthode est appelée par le répartiteur et permet d’effectuer les calculs qui lui sont demandés.

En mode sécurisé **compute** va essayer d'identifier le répartiteur auprès du serveur de noms en appelant la méthode à distance **authenticate**. Dans le cas où l’authentification a réussi elle effectue les calculs demandés. Dans le cas contraire elle renvoie la valeur *null* qui va alors être traitée de manière appropriée par le répartiteur.

En mode non sécurisé **compute** va tout de même effectuer les calculs qui sont demandés, mais pour simuler les erreurs va placer des résultats aléatoires à la place de ceux calculés. Pour cela elle génère un nombre aléatoire puis le compare au taux d’erreur entré.

Dans le cas où le serveur de calcul est interrompu intempestivement il va envoyé une exception *RemoteException* qui va être traitée par le répartiteur.

Scripts de lancement

Le README décrit les scripts de lancement. Il s'agit de charger dans tmux le fichier launch-everything-new.tmux. Ce fichier va ouvrir une fenêtre avec toutes les parties nécessaires.

Tests de performance

Par manque de temps les tests de performance n'ont pas pu être réalisés de manière rigoureuse dans la salle de TP. Cependant nous avons pu tout de même observer les tendances et temps approximatifs que ce soit pour le mode sécurisé et non sécurisé.

Mode sécurisé

En mode sécurisé, on divise la tâche selon le nombre de serveurs. On passe ces parties à des threads qui font faire le travail à chaque serveur. Chaque thread correspond à un serveur. Il passe le travail au serveur par "petites bouchées" de taille correspondant à la capacité du serveur.

Le temps pris dépend de la taille de la capacité du serveur. En effet, pour un même nombre d'opérations, si la capacité est plus petite, on doit briser le travail en plus de morceaux. Et chaque morceau demande une authentification et des transferts sur le réseau.

Pour des deux serveurs de capacité 3, le temps est d'environ 2.5 secondes en moyenne. Lorsqu'on réduit la capacité à 1, le temps monte à environ 3 secondes. Par contre, si on met la capacité plus grande, par exemple 100 par serveur (donc capacité illimitée) on ne fait pas beaucoup mieux que 2.5 secondes; on reste dans tous les cas au dessus de 2 secondes.

Nous pensons donc que le temps requis pour faire les opérations est beaucoup plus important que le temps pour les communications réseau et les authentifications. Comme toutes les tâches sont lancées en parallèle, l'élément limitant en terme de performances dans notre cas est le surcoût lié à la communication. Augmenter le nombre de serveurs ne diminue ainsi le temps de calcul que jusqu'à un certain point.

Mode non sécurisé

En mode non-sécurisé, les deux serveurs doivent tous les deux faire tout le travail et nous obtenons des temps d'opérations plus longs. Des temps d'environ 5 secondes et plus ont étés observés pour deux serveurs de capacité 3.

Changer la capacité de 3 à 10 à 100 ne fait pas une différence significative sur le temps de calcul. La baisser à 1 augmente le temps de traitement, mais de moindre mesure par rapport au temps total.