Polytechnique - Montréal

INF4410 – Systèmes répartis et infonuagique

TP2 – Services distribués et gestion des pannes

Auteurs :

Jérémy Wimsingues *1860682*,
Robin Royer *1860715*

10 novembre 2016

1 Choix de conception

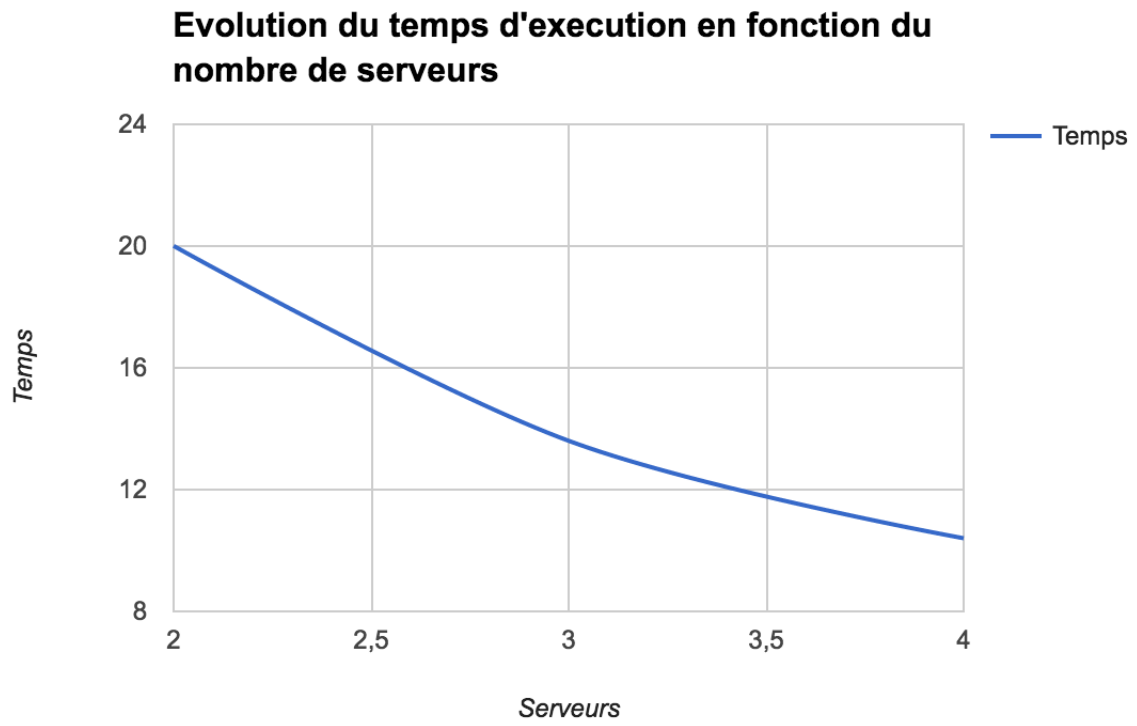
2 Tests de performances

2.1 Mode sécurisé

Nous avons utilisé pour ces tests le fichier **operations-1600** [1] qui contient 10 000 opérations unitaires. Voici les résultats des tests obtenus :

— Deux serveurs (q1 = 3 ; q2 = 6) : **20,006** secondes
— Trois serveurs (q1 = 3 ; q2 = 6 ; q3 = 9) : **13,604** secondes
— Quatre serveurs (q1 = 3 ; q2 = 6 ; q3 = 9 ; q4 = 12) : **10,403** secondes

Voici le graphique obtenu du temps d'exécution en fonction du nombre de serveurs.



Cette courbe présente une décroissante logarithmique en fonction du nombre de serveur de calcul utilisé. Nous remarquons donc que en mode "protégé", il est alors très intéressant de paralléliser les calculs. Cela présente un gain de temps significatif : le temps de calcul est divisé par 2 en multipliant le nombre de serveur par 2.

1. Fichier que nous avons créé nous mêmes qui est en faite 100 fois les calculs présents dans le fichier **operations-3216**.

2.2 Mode non-sécurisé

Nous avons utilisé pour ces tests le fichier **operations-1600** qui contient 10 000 opérations unitaires. Voici les résultats des tests obtenus sachant que dans tous les cas, les serveurs ont une valeur de q = 5 :

— Trois serveurs de bonne foi : **26,408** secondes
— Un serveur malicieux 50% du temps, deux autres serveurs de bonne foi : **75,217** secondes
— Deux serveurs malicieux 50% du temps, un autre serveur de bonne foi : **145,627** secondes

Nous remarquons sans surprise que les temps de calculs sont bien plus longs. Compte tenu de notre implémentation lorsque le mode non-sécurisé est activé nous vérifions tous les calculs sans savoir si les serveurs de calculs seront malicieux ou non. Cela explique notamment la différence entre le test réalisé avec trois serveurs de bonne foi avec q = 5 et le test similaire réalisé lors de la section précédente lorsque le mode était sécurisé. Nous avons ainsi 13 secondes de calculs supplémentaires pour le même fichier avec et sans mode sécurisé.

Le second test met en évidence le fait que nous ne sommes pas sûrs du résultat. 50% du temps un serveur renvoie un faux résultat que nous devons retraiter puis re-vérifier à nouveau car il est impossible de savoir s'il s'agit du calcul des opérations unitaires qui est faux ou s'il s'agit de la vérification. De ce fait, à chaque fois qu'un calcul est faux, nous devons relancer les calculs mais également la vérification. Ici la performance est vraiment atteinte car il faut plus d'une minute de calcul alors que avec trois serveurs nous convergions en seulement 13 secondes dans la section précédente.

Le cas de deux serveurs malicieux est intéressant car cela montre à quel point il va être difficile de converger vers une solution. Ici nous avons 50% des calculs qui seront faux 50% du temps pour chaque serveur. Cela augmentera considérablement le temps d'exécution. Le résultat est pire avec deux serveurs malicieux sur trois, il nous faudra plus de 145 secondes pour converger, soit une augmentation de 50% des temps de calculs par rapport au test précédent et une augmentation de 1000% par rapport au temps de calcul se la section précédente (145 secondes au lieu de 13).

D'un autre point de vue, cela prouve que si un attaquant (comprendre hacker) arrive à corrompre deux serveurs sur trois il est possible de grandement ralentir notre faculté à fournir des résultats justes. Aussi si ce dernier arrive à renvoyer la même valeur, alors il est possible qu'une opération soit validée alors qu'elle n'était pas juste. En effet, si nous validons un calcul erroné, autrement dit si le répartiteur demande de sommer 2 et 3 par exemple et que le premier serveur de calcul renvoie 0 et que le second valide ce résultat, alors le résultat sera validé par le répartiteur. La probabilité que cela se produise est alors de $1/4000 * 1/4000 = 1/16000$. Autrement dit nous avons une chance sur 0,00625% de chance que chaque calcul soit faux.

En conclusion de cela, nous pourrions donc avoir un "répartiteur intelligent" qui compte le nombre de vérifications erronées : si l'on a trop de vérifications qui échouent, nous sommes probablement en train de subir une attaque. Une des limitations d'une telle implémentation est l'attaque de plusieurs serveurs de calcul en simultané : l'idéal étant de toujours avoir plus de serveurs de bonne foi que de serveurs malicieux. Cependant, nous pourrions toujours avoir une condition de la sorte : si le taux de validation de vérification est inférieur à X (taux à fixer, exemple 95%), alors nous envoyons un courriel d'alerte à l'administrateur réseau qui devra se charger de vérifier l'authenticité des paquets transmis.

3 Question du TP

Le système distribué tel que présenté dans cet énoncé devrait être résilient aux pannes des serveurs de calcul. Cependant, le répartiteur demeure un maillon faible. Présentez une architecture qui permette d'améliorer la résilience du répartiteur. Quels sont les avantages et les inconvénients de votre solution ? Quels sont les scénarios qui causeraient quand même une panne du système ?

Dans le système distribué mis en place et expliqué dans la partie précédente, nous remarquons effectivement que le répartiteur constitue ce que l'on appelle un **point unique de défaillance** [2]. Ce type d'architecture, qu'il soit physique (exemples classiques : pare-feu, routeur) ou logiciel (exemples classiques : DNS, système d'annuaire type LDAP) est sensible lorsque l'élément en question est touché par une panne éventuelle.

Dans notre cas, le répartiteur est cet élément de l'architecture. S'il est atteint, tout le système est alors en échec. La notion de service, et plus précisément de continuité de l'activité, est une notion importante et il est alors bon de trouver des solutions pour limiter ces maillons faibles.

Pour résoudre les problématiques liées au SPOF, il n'y a pas énormément de solutions et nous allons expliquer pourquoi. L'une d'entre elles consiste à ignorer le problème. Autrement dit, suite à une analyse de risque par exemple, on

2. Single Point Of Failure ou SPOF en anglais.

considère qu'il est acceptable que le service soit interrompu (une heure, une journée, plusieurs jours ...), le temps de détecter la panne puis de relancer le système. Dans notre cas, par exemple, imaginons que notre répartiteur ne soit plus disponible (serveur en péril suite à un incendie, une inondation), alors il faut prendre le temps de le relancer, si besoin sur un autre serveur physique ou sur une autre machine virtuelle. Il faut ensuite régler certains paramètres de configuration liés à la nouvelle infrastructure (changement des adresses IP éventuelles, ainsi de suite), afin de relancer l'intégralité du système. Ce choix, bien qu'il puisse sembler naïf, est adopté par bon nombre de sociétés car il est parfois impossible de tout sécuriser comme on le voudrait, tout simplement par manque de budget. Nous parlons ici d'un répartiteur de calculs mais dans le cas d'un pare-feu, ou de licences de téléphonie IP ou d'annuaire, chaque licence est coûteuse pour une entreprise, sans compter l'infrastructure à déployer en conséquence (achat d'un serveur de secours par exemple).

Une autre solution, plus onéreuse, consiste à mettre en place de la redondance. Autrement dit, avoir plusieurs répartiteurs. De la même manière que nous constituons notre ferme (cluster en anglais) de serveurs de calculs, nous pouvons mettre en place une ferme de répartiteurs. Cette approche semble être logique après la lecture du paragraphe précédent. Autrement dit, nous doublons l'infrastructure et en cas de panne, le service redondé prend le relais. Cela est recommandé dans bons nombres de services critiques (par exemple dans la téléphonie IP comme Skype For Business, ou au niveau des serveurs Exchange pour les grosses sociétés utilisant la messagerie Miscrosoft). Mais souvent les coûts sont grands : détection de la panne, changement automatique des IP via des solutions de failover automatique, ainsi de suite. Cela sous-entend donc que la criticité du service soit grande, autrement dit que la continuité du service en question soit primordiale et nécessite donc un tel investissement.

Maintenant que ces deux approches ont été évoquées, quelle est la meilleure solution pour notre système ? La réponse est donc : tout dépend de la criticité du service. Si l'arrêt du répartiteur pendant plusieurs minutes entraîne des pertes de plusieurs milliers de dollars alors il semble légitime d'investir ces quelques milliers de dollars dans la redondance du répartiteur. Il faut donc mettre en place un second répartiteur qui communiquerait ainsi avec le premier. Si nous n'avons pas de réponse du répartiteur principal depuis plus de X secondes (TTL à fixer post analyse), alors ce second répartiteur prend le relais. Dans les système distribués, il est fréquent de voir ce genre de pratique. Mais cela suscite alors de nouvelles questions : comment récupérer les données stockées dans la mémoire de la JVM du répartiteur principal ? En effet, dans notre architecture nous utilisons des structures de données qui sont stockées dans la mémoire de la machine virtuelle qui contient le répartiteur. Si nous mettons en place un second répartiteur, le mieux étant sur une autre machine virtuelle et/ou sur un autre serveur physique, alors il faut pouvoir transférer la donnée d'un répartiteur à l'autre pour savoir où on en était dans les calculs. On commence donc ici à voir une notion de granularité : on pourrait avoir un second répartiteur prêt à l'emploi mais passif ou alors avoir deux répartiteurs actif/actif. Ce type d'architecture est aussi appelée "High-availability cluster". Mais si l'on investit dans ce type d'architecture, pourquoi ne pas faire également de la répartition de charge entre nos deux répartiteurs afin de mettre en place de l'actif/actif et d'avoir ainsi des performances accrues ? (load-balancing : nous envoyons un fichier de calculs sur chaque répartiteur qui se charge alors de le dispatcher sur les serveurs de calculs).

Finalement, nous nous rendons compte que la mise en place d'un tel service est compliqué, car si nous déplaçons le SPOF sur un autre aspect du système, le résultat n'en sera que le même : si le répartiteur de charge entre nos deux répartiteurs de calculs venait à tomber, quelle serait la conséquence ? Si le second répartiteur est sur la même VM ou sur le même serveur physique, le serveur physique devient un point individuel de défaillance. Si nous mettons un second répartiteur dans le même réseau mais que ces deux derniers ne sont reliés que par un câble ou par un unique routeur, alors ceux-ci deviennent des points individuels de défaillance à leur tour. Si nous utilisons le réseau pour effectuer des transferts des résultats courants d'un répartiteur vers un autre (afin de savoir où nous en étions dans les calculs), alors nous utiliserons la bande passante pour quelque chose qui ne se passera hypothétiquement jamais, ralentissant ainsi les échanges sur ce même réseau. Cela sous-entend donc la mise en place d'un lien redondé privé entre les deux répartiteurs, ce qui constitue un investissement supplémentaire.

Les exemples sont nombreux et on se retrouve ainsi dans une boucle sans fin qui nous pousse à redonder l'ensemble des éléments de notre système. Nous en revenons donc à la réponse évoquée plus haut : tout dépend de l'investissement que l'on souhaite effectuer compte tenu des pertes éventuelles en fonction de l'interruption de la continuité de notre service.

Sources utilisées : Wikipédia pour les articles sur le point individuel de défaillance[3]*, la redondance*[4]*, la répartition de charge*[5]*, ainsi que la notion de High availability cluster*[6].

3. https://fr.wikipedia.org/wiki/Point_individuel_de_d%C3%A9faillance
4. https://fr.wikipedia.org/wiki/Redondance
5. https://fr.wikipedia.org/wiki/R%C3%A9partition_de_charge
6. https://en.wikipedia.org/wiki/High-availability_cluster