# INF8480 - SYSTÈMES RÉPARTIS ET INFONUAGIQUE

# TRAVAIL PRATIQUE 2
# SERVICES DISTRIBUÉS ET GESTION DES PANNES

PRÉSENTÉ À :

HOUSSEM **DAOUD**

ANAS **BALBOUL**

PAR :

1773922, ÉTIENNE **ASSELIN**

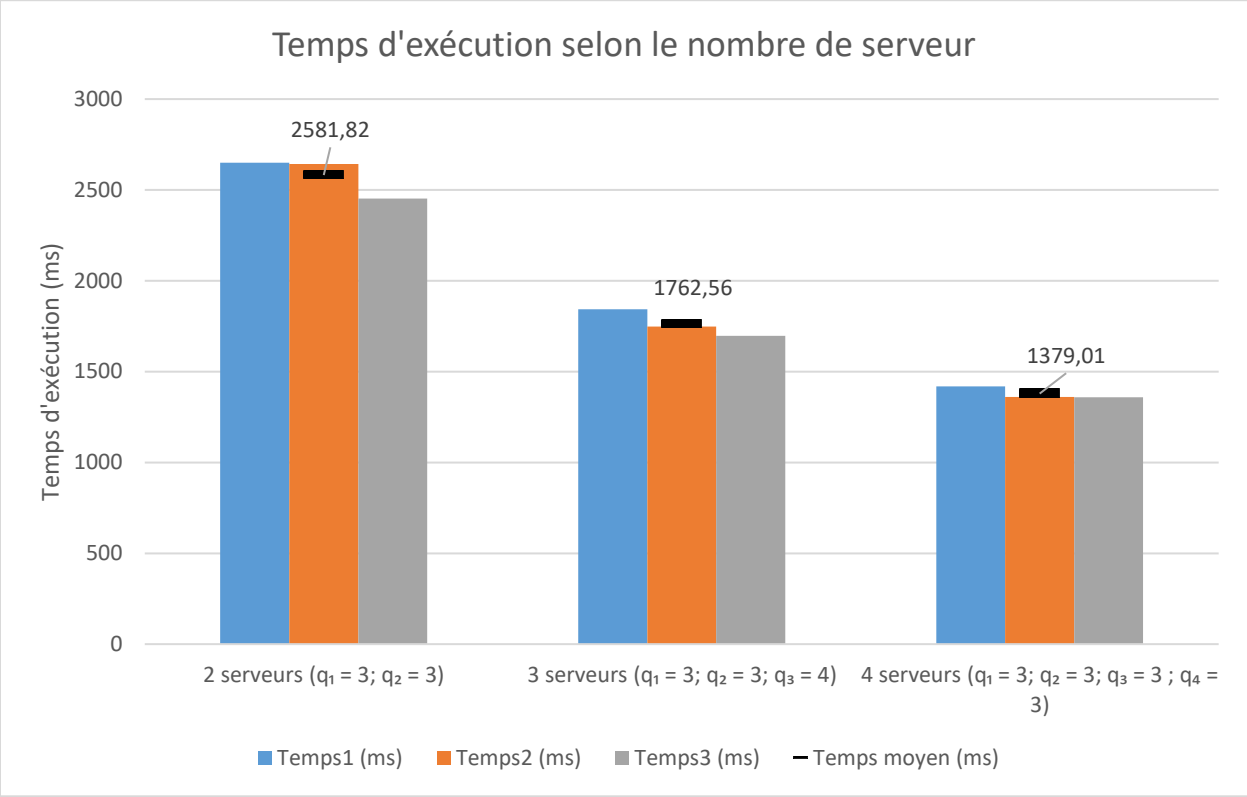
1744784, VINCENT **RODIER**

1734142, FRÉDÉRIC **BOUCHARD**

DATE : 20 MARS 2018

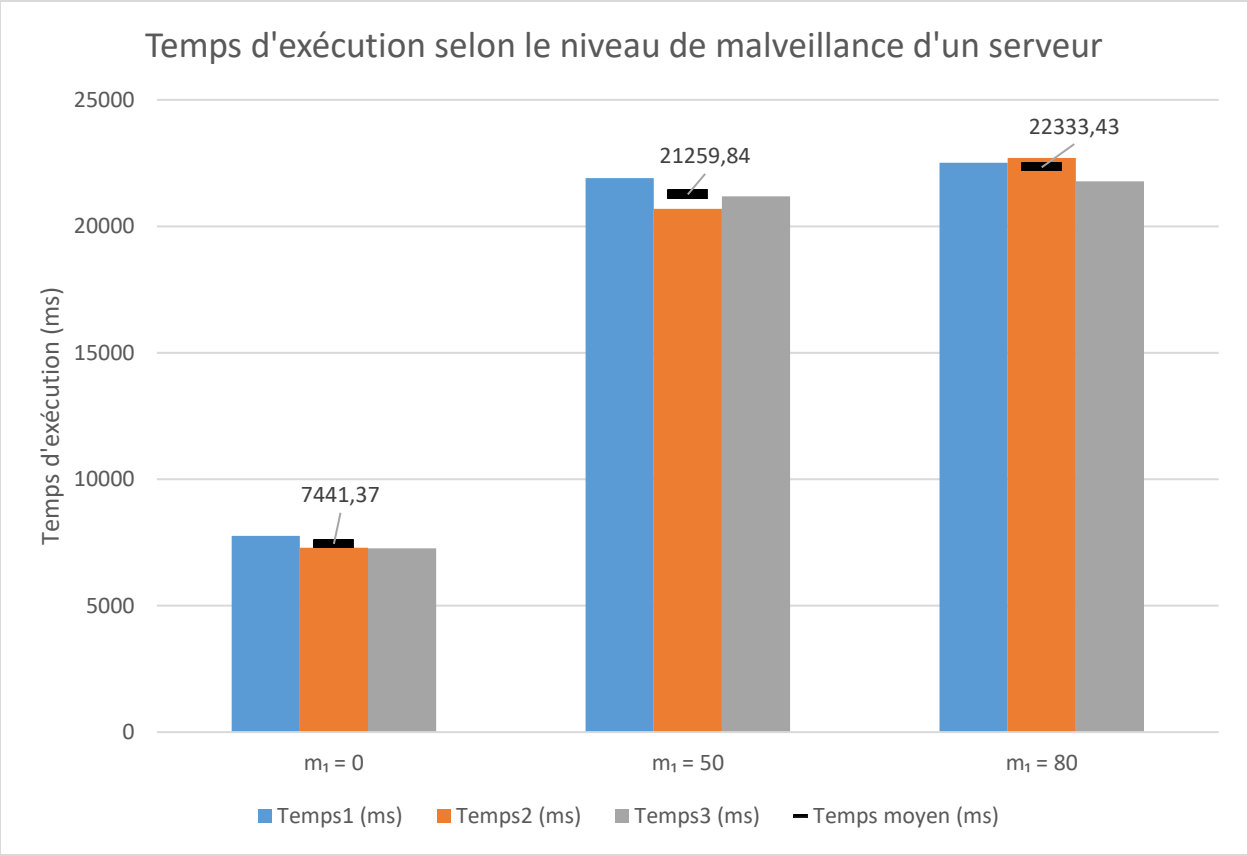
## Présentation des résultats

Graphique 1 : Temps d'exécution selon le nombre de serveur




En analysant le graphique ci-haut, on voit clairement une diminution du temps requis pour effectuer le nombre de calcul lorsque le nombre de serveur augmente. Par exemple, en doublant le nombre de serveur, on diminue le temps de près de 47%. De plus, il est intéressant de voir que le nombre de requête qu'un serveur peut accepter n'est pas négligeable. En ajoutant un serveur possédant un nombre supérieur d'acceptation de requête, le temps d'exécution diminue de près de 32%. À l'opposé, une autre configuration possédant un serveur supplémentaire mais un niveau d'acceptation de requête plus bas diminue le temps d'exécution de 22%.

Graphique 2 : Temps d'exécution selon le niveau de malveillance d'un serveur




Le graphique montre le temps requis pour effectuer une série de calcul si le répartiteur de charge s'assure que les résultats qu'il obtient concordent. Donc, le répartiteur questionne deux serveurs pour une série de calcul et compare les résultats. Si le résultat est identique, il assume que la valeur obtenue est bonne et continue à distribuer d'autres commandes. Dans le cas contraire, il requestionnera d'autres serveurs disponibles jusqu'à ce que le résultat soit identique. Dans le cas où tous les serveurs retournent le bon résultat, le temps d'exécution est assez faible (7441,37 ms). Cependant, dans les deux autres situations le temps augmente considérablement. Néanmoins, on s'aperçoit que le temps d'exécution plafonne peu importe le niveau de malveillance. Ceci s'explique de par la probabilité qu'une seule ligne d'exécution soit affectée par le niveau de malveillance tant vers une même valeur lorsqu'elle est supérieure à 50%.

## Question 1

Il est possible de constater que le répartiteur peut gérer des pannes de serveur. En effet, il peut redistribuer le bloc de calcul vers un autre serveur s'il constate qu'un serveur ne réponds plus, ou qu'il est tout simplement malicieux. Cependant, si le répartir tombe en panne lui-même, il n'existe aucune redondance dans le code afin d'assurer la continuité de la tâche. Dans l'architecture actuelle, si le répartiteur tombe en panne, les serveurs ne recevrons plus aucune tâche à effectuer.

Afin de pallier ce problème, nous allons utiliser la théorie derrière le LDAP. Tous les serveurs communiquent avec le LDAP. Dès lors, si un serveur n'a pas reçu de tâches depuis un certain moment, il pourrait envoyer un message au LDAP lui demandant si le répartiteur est toujours actif. Ainsi, le LDAP connaissant l'adresse IP du répartiteur pourrait *pinger* le répartiteur afin de valider que ce dernier est toujours actif. S'il ne reçoit pas de réponse ou bien qu'un message d'erreur soit capté lors de l'envoi du message, le LDAP pourra donc en conclure que le répartiteur est en panne. Le cas échéant, il pourrait instancier un nouveau répartiteur sur une autre machine distante. Ce nouveau répartiteur devra recommencer tous les calculs depuis le début afin de s'assurer qu'aucune donnée n'est manquante. Une autre façon de gérer le tout serait que tous les serveurs implémentent aussi les fonctions de répartiteur. De cette façon, le LDAP n'aurait qu'à assigner la tâche de répartiteur à l'un des serveurs actifs. Cette méthode serait problématique dans le cas où il n'y a qu'un minimum de serveurs disponibles pour faire les opérations. Dans ce cas, il faudrait alors que le LDAP crée une nouvelle instance du répartiteur et redirige les réponses des serveurs vers cette nouvelle machine. Le tout peut être très couteux en temps d'exécution. Recommencer tous les calculs pour un grand fichier peut prendre beaucoup de temps comme nous l'avons expérimenté avec un fichier de plus de 1000 opérations.

Malgré tout, il est possible de constater que tous les scénarios ne sont pas à l'abris de la panne. En effet, si le LDAP tombe en panne, aucune information ne pourra être transmise entre le répartiteur et les serveurs car c'est ce dernier qui connaît l'emplacement de tous les sous-systèmes du programme. Dans ce cas, cette nouvelle architecture ne couvrirait pas cette faille. Il faudrait alors penser à un système de redondance pour le LDAP lui-aussi, ce qui nous mène alors vers une solution qui nécessite toujours un système pour venir protéger une panne éventuelle.

## Question 2

En mode sécurisé, le répartiteur crée un *thread* pour chacun des serveurs qu'il connait. Ainsi, il a la responsabilité de répartir les opérations à exécuter à chacun de serveurs. Celles-ci se trouvent dans une liste d'opération non complétées qui est verrouillée par un *thread* chaque fois que celui-ci tente d'y accéder afin d'éviter la duplication d'opérations. Pour connaître le nombre d'opération que le serveur acceptera de faire lors de la prochaine requête, le répartiteur commence toujours par demander au serveur si celui-ci accepte une valeur cinq fois supérieure à son qi. Chaque fois que le répartiteur se fait refuser une demande, il diminue le facteur multiplicatif du qi de 1 jusqu'à acceptation. À la première demande, le taux de refus est donc de 80% ensuite de 60%, 40%, 20% et à la cinquième demande, le serveur accepte obligatoirement. Dans les faits, on observe rarement plus de deux ou trois refus par serveur et souvent une acceptation après seulement deux demandes.

En ce qui a trait aux pannes, le répartiteur reçoit une exception lorsqu'il fait un appel de fonction à un serveur. Lors d'un tel cas, le *thread* associé à ce serveur se termine après avoir remis les opérations qu'il avait réservé dans la liste d'opérations à faire.

En mode non-sécurisé, le fonctionnement est très similaire. Le répartiteur possède deux listes. La première contient les opérations à exécuter et la seconde contient tous les serveurs disponibles. Le répartiteur créer autant de *thread* qu'il y a de serveurs disponibles. Ensuite, chaque *thread* tente d'acquérir deux serveurs contenus dans la liste de serveurs disponibles en la verrouillant. Ensuite, le répartiteur prend la plus petite valeur de qi des deux serveurs et fait le même processus de demande d'acceptation que le mode. À ce moment, les deux serveurs exécutent les opérations sélectionnées de la même façon que pour la méthode sécurisée. Dans le cas où les serveurs s'entendent sur la valeur du calcul, le répartiteur prend en compte le résultat, remet les deux serveurs dans la liste de serveur disponible. Cependant, dans le cas où les serveurs ne retournent pas le même résultat, le répartiteur remet encore une fois les deux serveurs dans la liste de serveurs disponibles, puis il remet aussi les opérations dans la liste d'opérations à faire. Ces opérations seront exécutées tant et aussi longtemps que deux serveurs ne s'entendent pas sur le résultat. En cas de panne d'un serveur, le répartiteur reconnait le serveur fautif, remet le bon serveur dans la liste de serveurs disponibles et remet aussi les opérations dans la liste d'opérations à compléter.

Nous avons implémenté le système en mode sécurisé avec un thread par serveur puisque c'était la méthode la plus simple. Cette méthode nous a permis de tirer avantage du multithreading et de la fonctionnalité asynchrone des serveurs. Dans le cas du mode non-sécurisé, nous avons autant de *thread*

que de serveurs disponibles. Cette caractéristique nous permet d’être capable de continuer de travailler peu importe le nombre de serveur qui tombe en panne.