Performance

1. Scénario : Maintenir le temps de réponse à moins d’une milliseconde pour 98% du temps 2

2. Scénario: Augmenter la quantité maximale de transactions par secondes du système 5

3. Scénario : L’interface « front end access » peut difficilement traiter le volume de transactions reçu durant certaines pointes. 8

4. Scénario : Tous les utilisateurs se connectent en même temps après l’annonce de la vente de bombardier. 10

5. Scénario : Réduire le temps de traitement des requêtes 13

6. Scénario : Le logiciel boursier doit rapidement effectuer des transactions boursières 16

7. Scénario : Augmentation soudaine du nombre d’utilisateurs simultanés 18

8. Scénario : Nous désirons augmenter le nombre de transaction traités par seconde (plus de 20k/sec) 21

9. Scénario : Améliorer le temps d’enregistrement (persistance mémoire) 23

10. Scénario : Assurer qu’une transaction prenne moins de 1 ms peu importe le nombre de requête 28

11. Scénario : Une grande quantité de requête à l’externe via FIX et SAIL dans une très petites période augmentation ainsi la performance nécessaire du système. 31

# Scénario : Maintenir le temps de réponse à moins d’une milliseconde pour 98% du temps

| **objectifs d'affaires** | Maintenir le temps de réponse à moins d’une milliseconde pour 98% du temps |
| --- | --- |
| **Source** | Intervenant externe envoie une transaction sur l’interface du système |
| **Stimulus** | Transaction ou groupe de transactions |
| **Artéfact** | Le système |
| **Environnement** | Opération normale |
| **Réponse** | Gérer la transaction |
| **Mesure de la réponse** | Temps nécessaire pour traiter la transaction |
| **Questions** |  |

## Tactique 1 : Prioritize events

**Description**: Les transactions reçues sont classées selon leur priorité. S’il n’y a pas assez de ressources pour traiter les transactions de faible priorité, celles-ci sont ignorées.

**Justification**: Certaines transactions boursières peuvent être plus importante que d’autres. Par exemple, des gros achats ou ventes d’actions peuvent être plus important qu’une simple consultation.

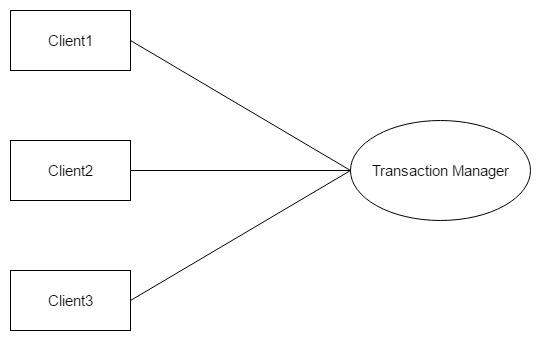
## Tactique 2 : Introduce concurrency

**Description**: Les ressources disponibles peuvent avoir plusieurs threads pour traiter les transactions en parallèle. Nous allons aussi introduire une *scheduling policy* de type *semantic importance* qui rejoint la Tactique 1 “Prioritize events”.

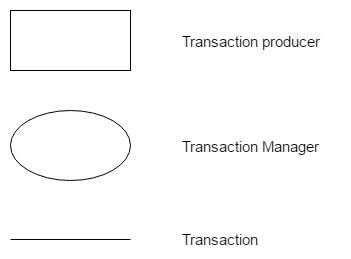
**Justification**: Pour compléter la première tactique, celle-ci permettrait d’avoir plusieurs threads pour diminuer les chances qu’une série de transactions de haute importance paralyse les transaction moins importante en attente.

## Vue architecturale (à l’extérieur de la classe)

### Diagramme



### Légende



### Texte de description du diagramme

Dans le diagramme, il y a trois clients externes qui demandent chacun au système de traiter une transaction. Les transactions passe pas le *Transaction Manager* qui lui sert à assigner une priorité de traitement aux transactions. C’est ce composant qui va aussi passer les transactions aux différents *threads* disponibles.

### Table de description des éléments du diagramme

| **Élément** | **Description** |
| --- | --- |
| Client | Envoi des requêtes au transaction manager. Il pourrait s’agir ici d’autre module ou de client physique comme dans un architecture client-serveur. |
| Transaction Manager | Module qui vas appliquer les différentes tactique pour maintenir la performance. |
| Transaction (Lien entre les clients et Transaction Manager) | Les clients envoient des transaction au systèmes en passant par le Transaction Manager |

### Texte décrivant la relation entre les éléments et les tactiques

Le Transaction Manager permet de définir quelle requête reçu de ces clients est la plus prioritaire (via la tactique *Prioritize event)*. Selon la priorité configuré il vas par la suite déterminer si il y a assez de ressource nécessaire pour gérer les transactions de moindre priorité. Par la suite selon l’importance des transactions le Transaction Manager vas déterminer sur quel thread envoyer la transaction (via la tactique *Introduce concurency*) afin de bien répartir la charge entre les différentes ressource et éviter qu’un thread soit surchargé (load balancing).

# Scénario: Augmenter la quantité maximale de transactions par secondes du système

| **Objectifs d'affaires** | Supporter environ 20k ordres de transactions/secondes |
| --- | --- |
| **Source** | Une requête externe |
| **Stimulus** | Suivre un pattern stochastique influencé par les habitudes de fréquentation de l’application par les utilisateurs. |
| **Artéfact** | Le système |
| **Environnement** | Peak load and overload |
| **Réponse** | Traite les transactions une après l’autre le plus rapidement possible |
| **Mesure de la réponse** | * Quantité maximale de transactions par minute supportées par le système. * La latence des réponses. * Le temps limite pour donner une réponse. * Le taux de ratage. |
| **Questions** | 1. Combien de processus en parallèle pouvons-nous supporter? 2. Quelle politique de planification devrait-on utiliser? 3. Est-ce que toutes les requêtes sont égales? |

**Tactique 1: Ajouter des ressources**

**Description:**

Comme le nom de la tactique indique, on ajoute des processeurs et de la mémoire pour réduire la latence. Ceci permet d’observer une amélioration de la performance immédiatement.

**Justification**:

L’addition de ressources allouée aux transactions va accélérer le rythme du traitement. En effet, pour supporter une plus grande charge de transactions nous avons besoin d’un plus grand nombre de ressources.

**Tactique 2: Introduire de la concurrence**

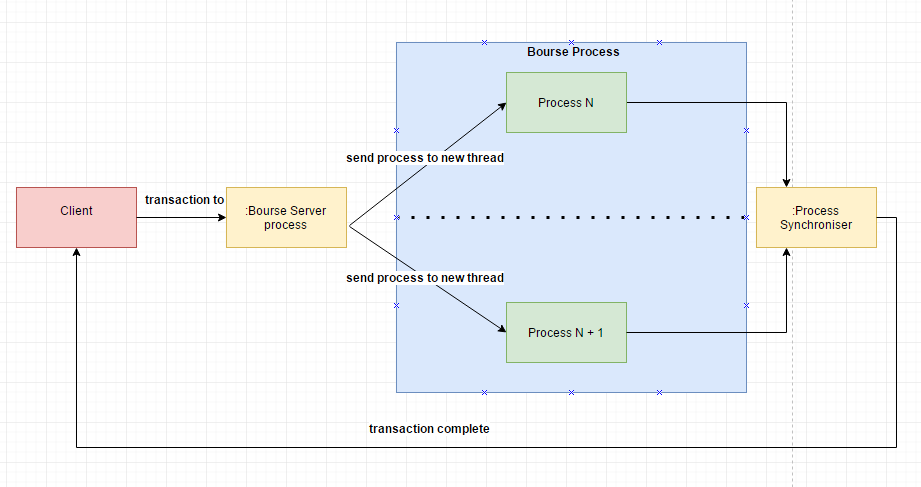
**Description:**

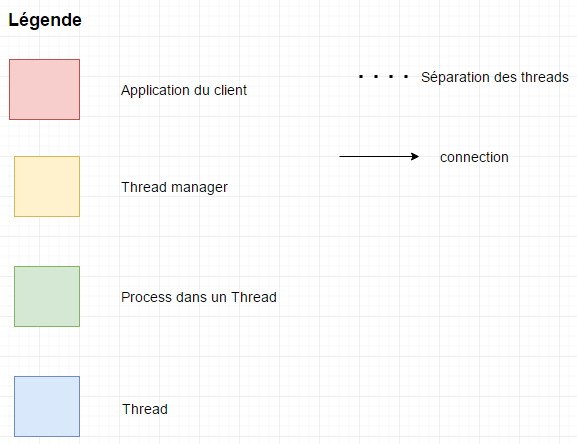
Introduire une concurrence dénote la capacité de traiter des événements différents en parallèle. Une fois que la concurrence a été introduite, des politiques de planification peuvent être utilisés. Par exemple, toutes les requêtes sont égales ou les requêtes ayant le temps de complétion le plus court sont traitées en premier.

**Justification**:

Plutôt que traiter des requêtes à tour de rôle (premier arrivé, premier servi), la concurrence permet de traiter plusieurs requêtes en parallèle, ce qui permettra d’augmenter le nombre de transactions par seconde et de réduire la latence.

### Diagramme et légende





### Texte de description du diagramme

Lorsqu’un client envoie une transaction au bourse server process, celui-ci va séparer la requête en plusieurs processus et qui va ensuite envoyer à un thread disponible pour faire la requête. Lorsque le processus est terminé, celui-ci va être envoyé dans le processus synchroniser où il synchronise les processus pour donner une réponse au client.

### Table de description des éléments du diagramme

| **Bourse server process** | Un système de traitement de requête |
| --- | --- |
| **Process N** | Un processus dans le thread |
| **Bourse process** | Un ensemble de différent threads |

# Scénario : L’interface « front end access » peut difficilement traiter le volume de transactions reçu durant certaines pointes.

| **objectifs d'affaires** | Tolérer des pointes de 20 000 ordres de transactions par seconde reçue sur le « front end access » (basé sur CU04) |
| --- | --- |
| **Source** | Réception des ordres via SAIL ou FIX |
| **Stimulus** | Réceptions d’un nombre X de transactions de manière stochastique (impossible de déterminer le nombre de transactions reçues à un moment Y de la journée) |
| **Artéfact** | L’API de « front end access » |
| **Environnement** | Fonctionnement normal |
| **Réponse** | Les transactions sont toutes traitées dans le délai prescrit. |
| **Mesure de la réponse** | Mesurer le temps de traitement d’une transaction, mesurer le nombre de transactions reçues par intervalle de temps, mesurer le nombre de transactions traitées dans un délai supérieur à la norme |
| **Questions** | 1. Quel est le délai de traitement souhaité pour une transaction? (en ms) 2. Que faire avec une transaction qui ne peut être traitée immédiatement? 3. Les transactions ont une priorité assignée ou sont-elles équivalente? 4. Est-ce que le temps de traitement d’une transaction varie selon le type de transaction? 5. Quelle est la capacité de traitement actuelle? |

## Tactique 1 : Manage ressources - Concurreny

**Description**: Les requêtes sont traitées en parallèle sur un serveur via l’utilisation de *thread* multiples pour une instance du service « front end access » afin de diminuer le temps d’attente de chaque requête.

**Justification**: Cela permet de traiter plus d’une requête à la fois plutôt que l’une à la suite de l’autre. Le nombre de requêtes traitées en parallèle dépendra de la capacité matérielle du serveur de gérer un nombre X de *thread*.

## Tactique 2 : Manage ressources – Multiple computation

**Description**: Plusieurs serveurs exécutant le même service et ordonnancés via un répartiteur de charge en amont pour assigner la tâche selon la charge de chaque instance. La répartition peut s’effectuer de façon circulaire (*round-robin*) ou selon le serveur étant le moins occupé. Le nombre de serveurs sera déterminé selon les capacités de chacun pour atteindre l’objectif d’affaires (ex : 2000 requêtes/serveur/sec \* 10 serveurs = 20 000 requêtes/sec du système dans l’ensemble).

**Justification**: Permets d’éviter un blocage de traitement lorsqu’un serveur a atteint le maximum de sa capacité de traitement.

## Tactique 3 : Manage ressources – Scheduling (ordonnancement)

**Description**: Assigner les ressources computationnelles selon un ordre de priorisation définit pour chaque requête reçue pour une instance du service.

**Justification**: Si une transaction est jugée plus prioritaire qu’une autre, celle-ci sera assignée en priorité à la ressource pour que son traitement soit effectué dans le délai prescrit.

## Vue architecturale (à l’extérieur de la classe)

### Diagramme

### Légende

### Texte de description du diagramme

Le diagramme représente une vue globale du parcours d’une transaction, de l’entrée jusqu’au traitement dans le *thread* du service. Le répartiteur de charge s’occupe de distribuer la requête vers le serveur/service jugé le plus apte à traiter la requête dans les délais prescrits, selon un algorithme choisi. À la réception de la requête, le service crée un nouveau *thread* pour effectuer le traitement requis, en parallèle aux autres en cours d’exécution. Un algorithme s’occupe d’allouer les ressources aux *threads* selon un ordre de priorité de la requête (contenu dans le *thread* à ce stade). Si une priorité n’est pas définie, l’ordre par défaut sera *FIFO*.

### Table de description des éléments du diagramme

| Transaction input | L’entrée d’un nombre aléatoire X de transactions dans le système |
| --- | --- |
| Load-Balancer | Distribue les requêtes vers le bon serveur/service selon un algorithme défini |
| Server | Contiens une instance du service (*process*) qui s’occupe du traitement de la requête. |
| Process | Permets de créer des *thread* pour chaque requête et d’allouer les ressources au *thread* selon un algorithme d’ordonnancement défini. |
| Thread | Effectue le traitement de la requête en utilisant les ressources allouées |

### Texte décrivant la relation entre les éléments et les tactiques

Le répartiteur de charge jumelé à plusieurs serveurs implémente plusieurs copies de la même ressource computationnelle pour s’assurer qu’il y a toujours un serveur capable de répondre à la requête de traitement de la transaction reçu dans le délai prescrit.

L’utilisation de plusieurs *threads* permet d’assurer un traitement concurrent de plusieurs requêtes. Cela permet de diminuer l’attente de chaque requête avant son traitement, augmentant ainsi le débit de traitement.

L’utilisation d’un algorithme d’ordonnancement dans le *process* gérant les *threads* permet d’allouer les ressources à la requête ayant la plus haute priorité afin qu’elle se termine dans les délais prescrits.

# Scénario : Tous les utilisateurs se connectent en même temps après l’annonce de la vente de bombardier.

| **objectifs d'affaires** | Le temps de réponse doit être inférieur à 1m sec pour 98% du temps. |
| --- | --- |
| **Source** | Externe au système (SAIL et/ou FIX) |
| **Stimulus** | Un nombre incroyable de requêtes est fait au système pour faire ou récupérer des transactions. (Événement stochastique) |
| **Artéfact** | Serveur, base de données |
| **Environnement** | Peak load |
| **Réponse** | * Réorganisation des tâches à effectuer. * Réallocation des ressources. |
| **Mesure de la réponse** | * Quantité de ressources libérées. * Nombre de requêtes faites par minute. |
| **Questions** | 1. Quel est le maximum de connexions faites en même temps en périodes de pointe? 2. Quelle est la quantité moyenne de requêtes qu’il y a en même temps? 3. Quelle est la quantité de serveurs à notre disposition? 4. Quelle est la différence de temps de traitement entre les requêtes faites avec SAIL et les requêtes faites avec FIX? 5. Lequel des deux systèmes est le plus prioritaire ? |

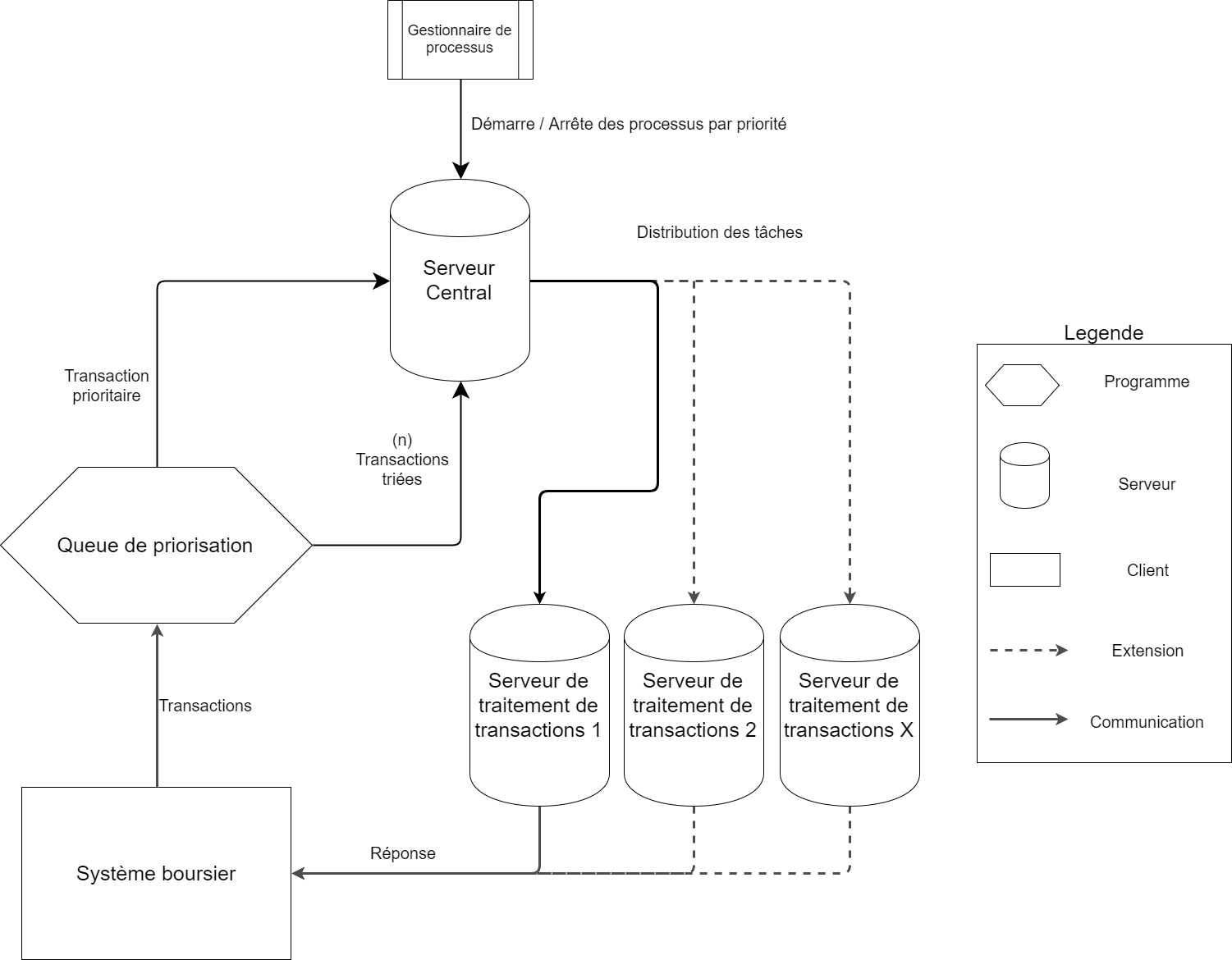
## Tactique 1 : Prioriser les événements

**Description** : Les événements ont des priorités différentes en fonctions que la criticité des transactions demandées. Plus la requête est urgente, plus la priorité sera élevée et elle sera traitée avant les moins urgentes.

**Justification** : Puisqu’il peut y avoir un grand nombre de requêtes reçues par le système en même temps, on a intérêt à mettre des priorités afin de les traiter de la plus prioritaire à la moins importante.

## Tactique 2 : Réduire «overhead»

**Description**: Des plugins et des processus annexes roulent sur le serveur. Nous les réduisons ou les arrêtons pour libérer des ressources.

**Justification**: Certains plugins de sauvegarde automatique et de surveillance peuvent être temporairement désactivés afin de pouvoir traiter plus de connexion au système. Ils seront réactivés ultérieurement. Nous optimisons ainsi les ressources existantes.

## Tactique 3 : Introduire de la concurrence

**Description**: On peut assigner différentes tâches à plus d’une machine (serveur) afin qu’elles soient traitées rapidement en parallèle.

**Justification**: Puisque le nombre de connexions nécessite plus de puissance pour un court délai, l’emprunt d’un second serveur d’appoint permet de séparer la charge et équilibre le stress équitablement. Par exemple, on peut séparer le traitement des requêtes faites à partir de SAIL du traitement des requêtes provenant de FIX.

## Vue architecturale (à l’extérieur de la classe)

### Diagramme

### Texte de description du diagramme

Ici, la tactique de priorisation des événements est modélisée par une queue de priorisation dont les entrants sont des transactions et les sortants sont une liste de transactions triée selon leur priorité.

Ces transactions sont passées au serveur central et ensuite distribuées aux serveurs de traitement de transactions en parallèle. Ces serveurs en parallèle représentent la tactique d’introduction de la concurrence. Après le traitement d’une transaction, une réponse est renvoyée vers le système boursier pour signaler la complétion de l’événement.

### Table de description des éléments du diagramme

| Élément | Description |
| --- | --- |
| Système boursier | Intermédiaire de communication entre le client et le serveur. |
| Transaction | Transaction boursière générique. |
| Queue de priorisation | Programme permettant de trier les transactions seulement un algorithme de détermination de priorité. |
| Serveur central | Serveur physique représentant le système central de communication entre les autres systèmes (base de données, serveurs de traitement de transactions, système boursier, etc.) |
| Serveur de traitement de transactions | Serveurs servant spécifiquement à traiter les transactions données. |
| Réponse | Résultat de la transaction envoyée. |

### Texte décrivant la relation entre les éléments et les tactiques

Le client communique avec le serveur central par intérim du système boursier. Il envoie une transaction générique, qui se fait trier parmi les transactions dans la queue actuelle et ensuite réacheminée vers le serveur central.

Par la suite, le serveur central distribue les transactions triées selon leurs priorités vers les serveurs de traitement de transactions en parallèle. Enfin, les résultats de ces transactions sont ensuite renvoyés vers le système boursier en tant que réponse au client.

# Scénario : Réduire le temps de traitement des requêtes

| **Objectifs d'affaires** | Réduire le temps de traitement des requêtes |
| --- | --- |
| **Source** | Développeur |
| **Stimulus** | Nouvelle transaction à traiter |
| **Artéfact** |  |
| **Environnement** | ~~Temps d'exécution~~ (mode opérationnel : normal) |
| **Réponse** | * ~~Donner un temps d’exécution pour chaque transaction réalisée~~ |
| **Mesure de la \_ réponse** | * Nombre de transactions par secondes * Nombre maximal de transaction par secondes |
| **Questions** | 1. Quel algorithme doit-on implémenter pour augmenter la rapidité? 2. Est-ce que la réduction du temps d’exécution est assez significative pour faire une différence? 3. Combien de temps avons nous pour atteindre l’objectif d’affaire? |

## Tactique 1 : Augmenter l’efficacité du calcul

## Description: Améliorer les algorithmes utilisés pour réduire la latence.

**Justification:** Réduire le nombre de ressources utilisés par le processeur et/ou le disque en améliorant les algorithmes utilisés dans les zones critiques de l’application, afin de diminuer la latence.

**Tactique 2 : Introduire la concurrence**

**Description:** Traiter les requêtes en parallèle, afin de réduire le “temps bloqué”.

**Justification:** Traiter les requêtes sur plusieurs threads exécutés en parallèle et équilibrer les charges de ceux-ci vont permettre de réduire le “temps bloqué”, ainsi cela va diminuer la latence de l’application.

**Tactique 3 : Augmenter les ressources**

**Description:** Augmenter le nombre de ressources disponibles, afin de réduire la latence.

**Justification:** Changer les processeurs pour des plus rapides, mettre des processeurs et/ou de la mémoire vive supplémentaires, ou augmenter la vitesse du réseau peut réduire la latence.

## Vue architecturale (à l’extérieur de la classe)

### Diagramme

### Publish and subscribe



### Légende

Une boîte équivaut à une composante

La ligne jaune est le bus de communication

Les ligne représentent des émissions d’événements

### Texte de description du diagramme

Dans le diagramme ci-haut, un client qui utilise le frontend access fait un transaction. Le module Frontend Access créé un évènement qui sera transmise sur le queue manager. Ce dernier enverra l’événement à tous les composants qui sont abonnés à l'événement.

### Table de description des éléments du diagramme

**System Monitoring** : Composante qui permet de s’assurer de la rapidité et de la disponibilité des différentes composantes et opérations.

**Market Data Dissemination**: Composante qui permet de publier des offre d’achat ou de vente à travers les différents marchés boursiers.

**Frontend access** : Composante qui permet au client de faire une transaction au système boursier.

### Texte décrivant la relation entre les éléments et les tactiques

Les tactiques seront appliquées au système lui-même et non aux composantes. Ainsi, on va **augmenter l’efficacité du calcul** de certains algorithmes critiques et **introduire la concurrence** dans notre système.

Par la suite, s’il y a toujours des problèmes de rapidité, on va **augmenter les ressources** de notre système, afin d’améliorer les performances de celui-ci.

Enfin, la composante *System Monitoring* va nous montrer si les performances sont bonnes.

# Scénario : Le logiciel boursier doit rapidement effectuer des transactions boursières

| **Objectifs d'affaires** | Temps de réponse inférieur à 1m sec pour 98% du temps |
| --- | --- |
| **Source** | Utilisateur(s) |
| **Stimulus** | Éffectue une transaction |
| **Artéfact** | Système, canaux de communication |
| **Environnement** | Fonctionnement normal |
| **Réponse** | Transaction traitée |
| **Mesure de la réponse** | Latence |
| **Questions** | 1) Combien de coeurs physiques et logiques ont chaque serveur qui traite les requêtes des clients?  2) Quel traitement doit être terminé avant d’envoyer une réponse au client? |

## Tactique 1 : Ajouter des ressources

**Description**: Ajouter des serveurs pour mieux répondre à la demande

**Justification**: Plus de ressources permettent logiquement de traiter plus de requêtes en même temps, donc plus rapidement.

## Tactique 2 : Introduire la concurrence

**Description**: Ajouter de la concurrence dans le code là ou c’est possible

**Justification**: En ayant plusieurs threads travaillant à une même tâche, on augmente la vitesse d’exécution d’une tâche.

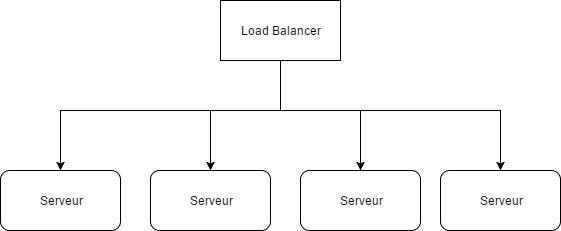
## Tactique 3 : Prioriser les événements

**Description**: Décaler le traitement des événements qui sont moins prioritaires

**Justification**: En ne traitant que ce qui est urgent, on réduit le temps de traitement pour ce qui est important. Par exemple, l’output d’un log peut être effectué après avoir répondu à une requête puisque c’est une tâche relativement lourde et qui n’est pas urgente.

## Vue architecturale (à l’extérieur de la classe)

### Diagramme



### Légende

Rectangle = Load Balancer

Rectangle arrondi = Serveur

### Texte de description du diagramme

Le load balancer reçoit les requêtes de l’extérieur et les distribue aux différents serveurs selon leur charge de travail et leur capacité. Le serveur est ensuite en mesure de renvoyer sa réponse au load balancer qui peut la relayer au client directement.

### Table de description des éléments du diagramme

Le load balancer reçoit les requêtes et les distribuent aux différents serveurs.

Les serveurs traitent les requêtes.

### Texte décrivant la relation entre les éléments et les tactiques

La présence d’un load balancer nous permet d’avoir plusieurs ressources en parallèles afin de balancer la charge de travail. De plus, chaque serveur utilise la concurrence dans le code afin de traiter les requêtes le plus rapidement possible. Finalement, le code de chaque serveur utilise une file de priorité afin de traiter les événements les plus importants en premier.

# Scénario : Augmentation soudaine du nombre d’utilisateurs simultanés

| **objectifs d'affaires** | Le temps de réponse doit être inférieur à 1ms pour 98% du temps. |
| --- | --- |
| **Source** | Le serveur qui reçoit les demandes |
| **Stimulus** | Nombre de requêtes soudainement élevé |
| **Artéfact** | Serveur de gestion des requêtes |
| **Environnement** | Surcharge |
| **Réponse** | Augmenter le nombre de serveurs de traitement dynamiquement |
| **Mesure de la réponse** | Le nombre de requêtes maximal que le serveur peut gérer sans ralentir.  La cadence de traitement est ralentie par la surcharge des requêtes |
| **Questions** | 1. À partir de quel nombre de requêtes le temps de réponse dépasse 1ms ? 2. Combien de requêtes prévues au maximum ? |

## Tactique 1 : Disponibilité des ressources

**Description**: Identifier les endroits où l’indisponibilité des ressources affecterait la capacité de traitement

**Justification**: Nous pouvons identifier que, si le serveur principal de traitement est indisponible ou surchargé, la cadence de traitement (throughput) va être fortement réduite.

## Tactique 2 : Augmenter les ressources

**Description**: Augmentation de la qualité des serveurs.

**Justification**: Remplacer les serveurs actuels par d’autres serveurs de meilleure qualité afin d’augmenter le nombre de requêtes maximal que chaque serveur peut gérer.

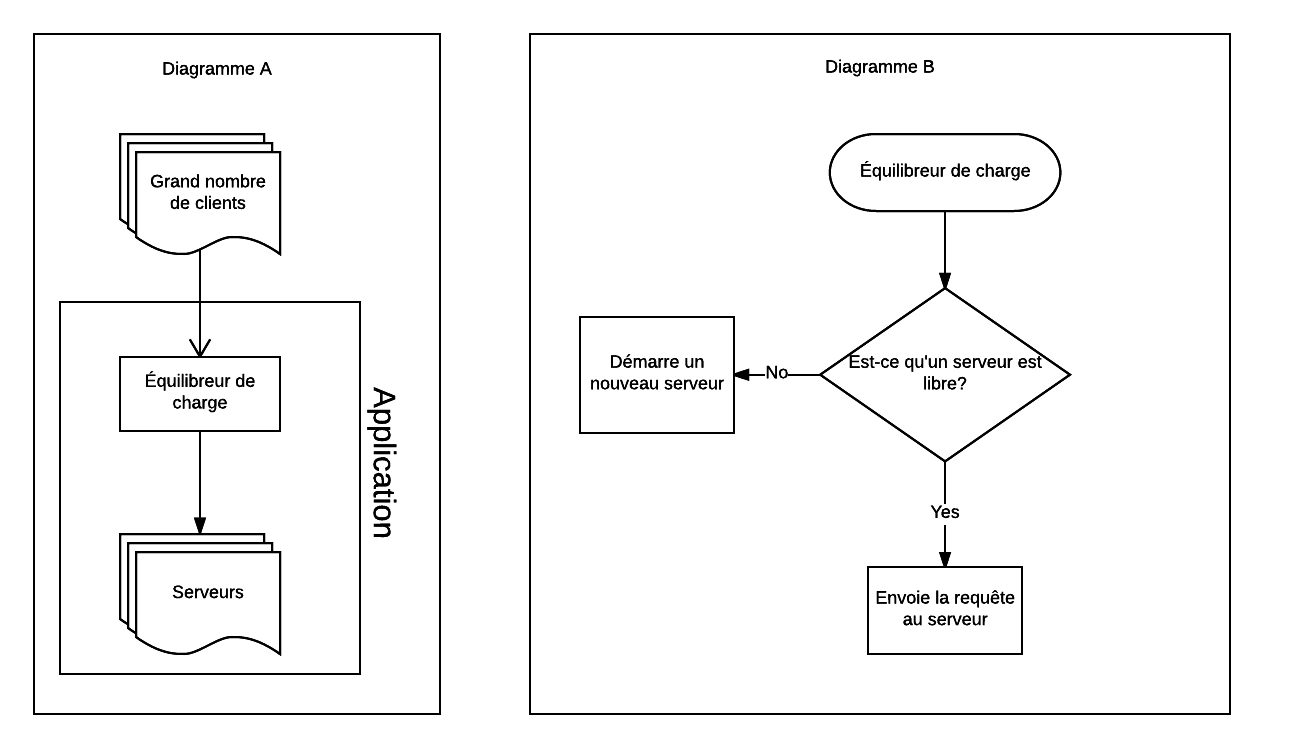
## Tactique 3 : Maintenir plusieurs copies d’unités de calcul

**Description**: Augmenter le nombre de serveurs gérant les requêtes.

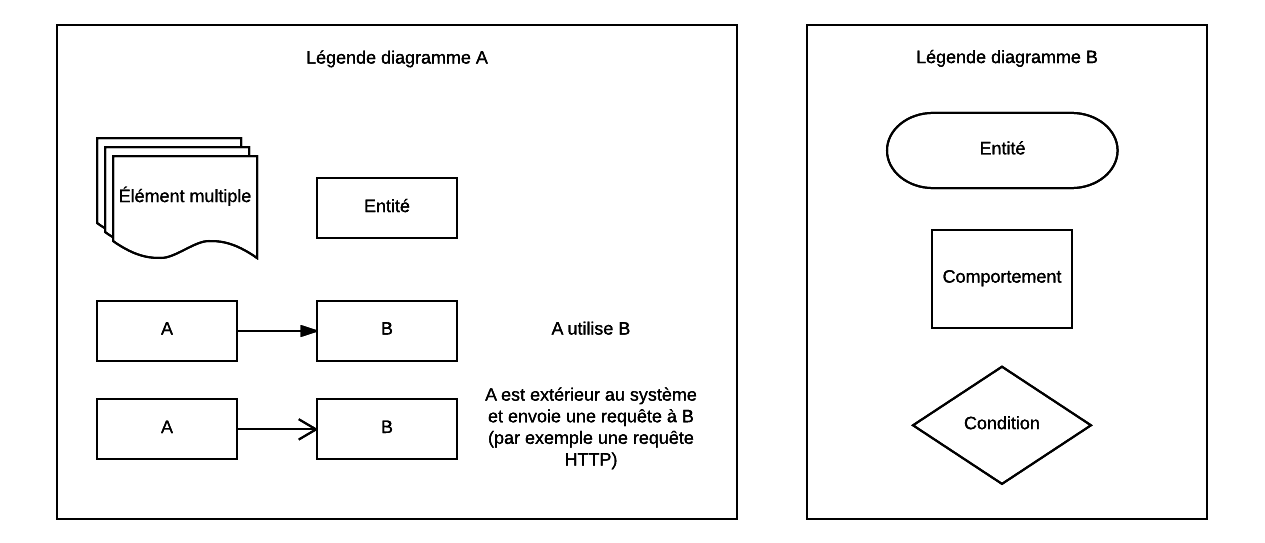
**Justification**: Augmenter le nombre de serveur diminue la charge de travail que chaque serveur doit effectuer.

## Vue architecturale (à l’extérieur de la classe)

### Diagramme



### Légende



### Texte de description du diagramme

Le diagramme A montre un grand nombre de clients envoyant des requêtes vers l’application. L’équilibreur de charge se chargera de diriger les requêtes vers un serveur prêt pour la traiter. De cette façon il sera possible d’augmenter le nombre de serveur (Maintenir plusieurs copies d’unités de calcul) ainsi que les améliorer (acheter des serveurs plus performant donc augmentation des ressources) car nous aurons identifier l’endroit où une surcharge de requête pourrait causer une indisponibilité des ressources.

Le diagramme B montre un peu plus en détail comment l’équilibreur de charge réagira quand il recevra une requête à traiter.

### Table de description des éléments du diagramme

| Clients | Les utilisateurs/poste physiques qui vont envoyer des requêtes à l’application |
| --- | --- |
| Application | Le côté serveur de l’application qui recevra les requêtes des clients et devra les traiter |
| Equilibreur de charge | Entité dans l’application qui va attribuer une requête à un serveur libre et prêt à y répondre |
| Serveurs | Les divers serveurs qui interagiront avec les donnés de l’application pour répondre aux requêtes des clients. |

### Texte décrivant la relation entre les éléments et les tactiques

L’équilibreur de charge va s’occuper d’attribuer une requête à un serveur qui est prêt à la traiter, de cette façon la réponse sera rapide. On peut donc dire que l’équilibreur de charge est né de l’application de la tactique visant à Maintenir plusieurs copies d’unités de calcul (les serveurs étant les unités de calcul). L’équilibreur de charge est aussi un point d’entrée des requêtes au niveau de l’application donc il est aussi une application de la tactique visant la disponibilité des ressources. La tactique “Augmenter les ressources” sera également appliquée en achetant plusieurs serveurs plus performants pour répondre au grand nombre de requêtes.

# Scénario : Nous désirons augmenter le nombre de transactions traitées par seconde (plus de 20k/sec)

| **Objectif d'affaires** | Supporter un plus grand nombre d’ordres de transaction par seconde. |
| --- | --- |
| **Source** | Clients externes qui utilisent le système transactionnel |
| **Stimulus** | Événement stochastique (Traitement des transactions à un certain moment de la journée) |
| **Artéfact** | Market Operation Control |
| **Environnement** | Achalandage maximal (Peak load) |
| **Réponse** | Plus de transactions sont traitées par seconde |
| **Mesure de la réponse** | Différence entre le débit de transaction initial (20k/sec) et le nouveau débit de transaction |
| **Questions** | 1. C’est quoi le niveau d’achalandage maximal? 2. Quelle est la distribution statistique de l’achalandage? |

## Tactique 1 : Increase Concurrency

Augmenter le nombre de machines de traitement des ordres de transaction. Cette augmentation permettra de répartir la charge de travail entre les serveurs et ainsi de traiter plus de transactions par seconde grâce au parallélisme.

## Tactique 2 : Increase Resource Efficiency

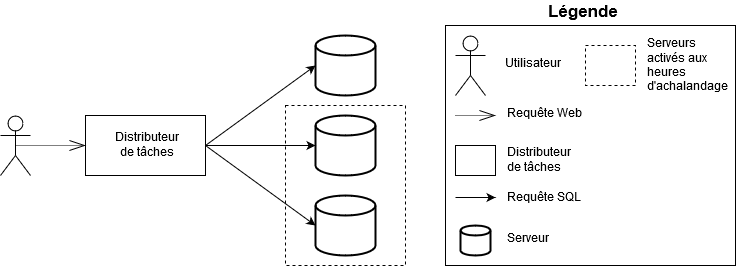
Chaque ressource devrait être utilisée de façon optimale et à son potentiel maximum sans compromettre la surcharge de travail/effort par ressource. (I.e.: utiliser le système de traitement à un certain pourcentage sans que ça dépasse une limite critique où nous pouvons augmenter le risque de panne/overload)

## Tactique 3 : Schedule Resources

Les ressources disponibles pour supporter le traitement des transactions peuvent être gérées selon la période de la journée. C’est-à-dire que lors des périodes d’achalandage élevé, nous pouvons mettre en place plus de ressources. Lorsque le taux d’achalandage n’est pas trop élevé il n’est pas nécessaire d’avoir toutes les ressources en marche.

## Vue architecturale

### Diagramme



### Texte de description du diagramme

Le diagramme ci-haut présente la configuration pour supporter un niveau élevé d’**utilisateurs** sans affecter la performance du système. Les **serveurs** dans le pointillés sont les **serveurs de support**, activés seulement aux heures d’achalandage élevé et le **distributeur** de tâches est l’élément clé de cette configuration puisqu’il doit rediriger les demandes **utilisateurs** aux **serveurs** disponibles, en respectant la **limite de charge** établie et assurer l’activation des **serveurs de support** aux heures d’achalandage.

### Table de description des éléments du diagramme

| **Élément** | **Description** |
| --- | --- |
| Utilisateur | **Utilisateur** du système. |
| Requête Web | **Demande de l’utilisateur** envoyée au système via une interface Web. |
| Distributeur de tâches | Redirige les **demandes utilisateurs** vers un **serveur** disponible, tout en respectant la **limite de charge** établie, et assure l’activation de **serveur supplémentaires** aux heures d’achalandage élevé. |
| Requête SQL | **Requête** du système pour le **serveur**. |
| Serveur | Conserve et envoie les **données demandées**. |
| Encadré pointillé | Les **serveurs** à l’intérieur du pointillé sont **serveurs** inactifs par défaut et activés lors de périodes d’achalandage élevé. |

### Texte décrivant la relation entre les éléments et les tactiques

D’abord, les **serveurs de support** servent la tactique d’augmentation de concurrence (*Increase Concurrency*) puisqu’ils augmentent directement le nombre de machine pour augmenter la performance du système. Le **distributeur de tâches** est quant à lui l’élément clé les deux autres tactiques, *Increase Resource Efficiency* et *Schedule Resources*. Il est utilisé dans la première tactique en distribuant les **requêtes** aux différents **serveurs** pour respecter la **limite de charge** de chacun et dans la seconde tactique en activant ou désactivant les **serveurs de support** selon les périodes d’achalandage du système.

# Scénario : Améliorer le temps d’enregistrement (persistance mémoire)

| Objectifs d'affaires | Réduire le temps entre les procédures d’enregistrement |
| --- | --- |
| Source | Le système interne |
| Stimulus | L’enregistrement sur le serveur. Stochastique (événements prévisibles). |
| Artéfact | Le serveur |
| Environnement | Opération normale et en période de pointe |
| Réponse | Les enregistrements sont effectués efficacement. |
| Mesure de la réponse | Latence pour le traitement des enregistrements (ms) |
| Questions | 1. Quel matériel physique est-il possible de changer ? 2. Quelles méthodes de sauvegarde pourraient remplacer la base de données actuelle ? 3. Quelle quantité de données peut être enregistrée par secondes ? |

# 

## Tactiques

## Tactique 1 : Limiter le temps de réponse - Limit event response

Description : Il arrive parfois que des événements arrivent trop rapidement pour être correctement reçus, analysés et traités. C’est la raison pour laquelle on favoriserait l’envoi de certains événements dans une file jusqu’à ce qu’ils puissent être traités. Il faut toutefois s’assurer que les files peuvent supporter les pires scénarios.

Justification : Via cette tactique, on tente de contrôler la demande de ressources en limitant ce qui est traité lors que les événements surviennent à de grandes vitesses de traitement. Ainsi, des éléments qui peuvent ralentir le système sont stockés dans des files qui seront vidées lorsque les éléments peuvent être traités. On contrôle ainsi l’entrée des événements du système en optimisant leur traitement et leur ordre de passage.

## Tactique 2 : Augmenter l’efficacité des ressources - Increase resource efficiency

Description : Optimiser le code afin que les utilisateurs se connectent de façon plus rapide.

Justification : Modifier les algorithmes présents d’authentifications. Les données des utilisateurs pourraient être cryptées et enregistrées en mémoire sur le serveur. Ainsi, les connexions des utilisateurs se feraient beaucoup plus rapidement.

## Tactique 3 : Augmenter les ressources - Increase resources

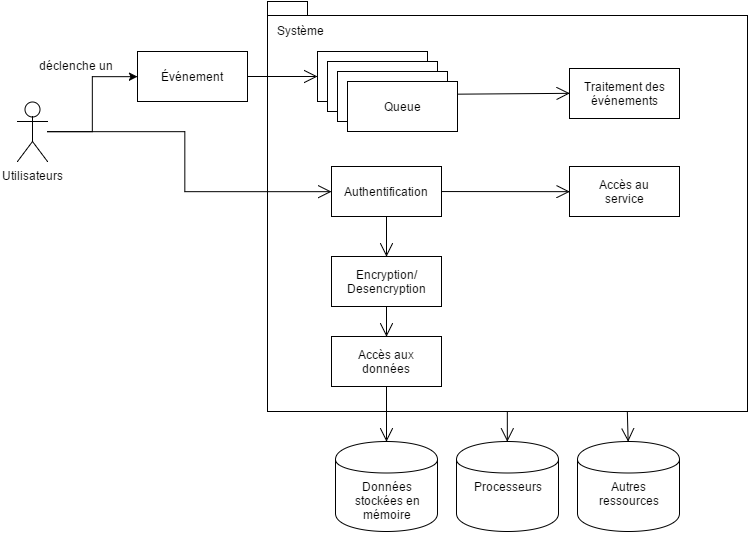
Description : Augmenter les performances matérielles du serveur. Pour y parvenir; ajout de mémoire vive, et augmentation de la puissance des processeurs.

Justification : Il faut que le serveur soit en mesure d’effectuer une réponse immédiate à tout moment, même lors d’un achalandage majeur. Pour ce faire, la conservation des données en mémoire vive permet de répondre à ce besoin. Considérant la quantité de requêtes reçues, l’augmentation de la quantité de mémoire résoudra le problème.

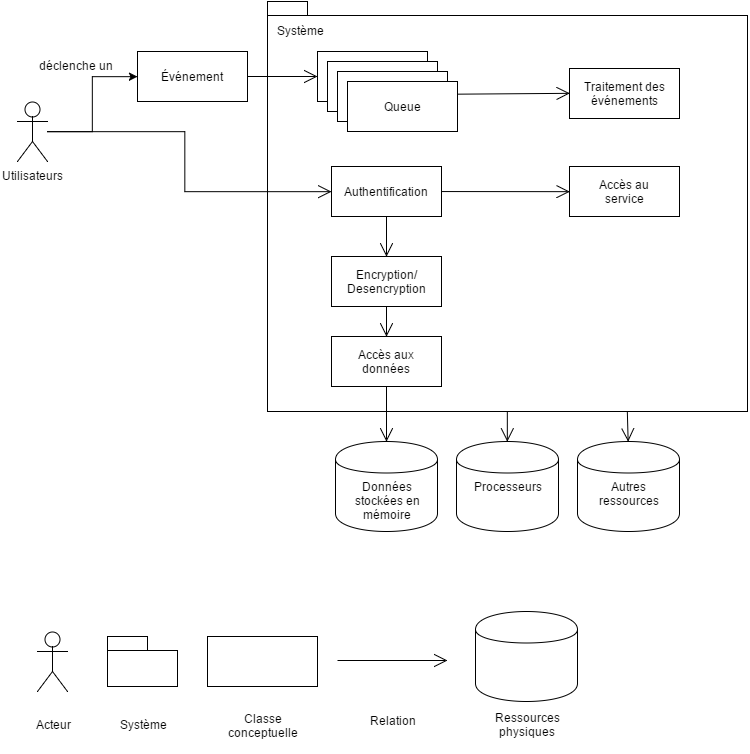
## 

## Vue architecturale

### Diagramme



### Légende



### Texte de description du diagramme

Nous utilisons un système en deux étapes qu’on peut observer sur le diagramme ci-dessus. On y passe la gestion d’événements ainsi que la gestion des actions d’un usager en lien un marché boursier. D’abord, lorsqu’un événement est déclenché, soit une requête de la part d’un utilisateur, et si le serveur est surchargé lors du traitement, celui-ci est placé dans la Queue de messages. À l’aide de cette file, nous sommes en mesure de gérer une surcharge d’événements lorsque le système est en mesure de le faire, donc quand il est disponible. Par la suite, nous avons l’étape d’authentification. Cette dernière permet d’avoir accès à divers services du système boursier. Après l’encryption ou la période de décryptage, on permet l’accès aux données stockées en mémoire en lien avec le système boursier. On y aperçoit aussi les diverses ressources physiques permettant de faire opérer le système efficacement. Plus ces composantes sont performantes et en grande quantité, plus le système sera efficace et performant. Cette dite performance peut aussi être améliorée par des algorithmes optimisés insérés dans le code source du système.

### Table de description des éléments du diagramme

| **Élément** | **Responsabilité** |
| --- | --- |
| Événement | Action effectuée par un utilisateur |
| Queue | Enregistre les événements |
| Traitement des événements | Traite les événements |
| Authentification | Authentifie l’utilisateur |
| Données des utilisateurs en mémoire | Conteneur des données |
| Processeurs | Composante physique utilisée pour les calculs |
| Autres ressources | Disque dur, CPU , mémoire vive, etc.. |
| Données stockées en mémoire | Contenir les informations globales à plus long terme |

### 

### 

### Table décrivant la relation entre les éléments et les tactiques

| **Tactique** | **Description** |
| --- | --- |
| Limit event response | Un utilisateur déclenche un événement qui est envoyé au système. Le système place l’événement dans une file d’attente pour que la requête soit ensuite traitée. |
| Increase resource efficiency | Un utilisateur tente de se connecter au système en utilisant ses identifiants. Le système utilise une base de données stockée en mémoire cache afin d'accélérer le processus d’identification. Ce cache contient les informations nécessaires afin de valider une authentification. L’utilisateur a ensuite accès au service si l’accès est autorisé. |
| Increase resources | Le système est utilisé sur un serveur contenant des ressources physiques très performantes. Cela permet aux requêtes de s’effectuer plus rapidement. |

# Scénario : Assurer qu’une transaction prenne moins de 1 ms peu importe le nombre de requête

| **objectifs d'affaires** | Faire des transactions dans un délai raisonnable |
| --- | --- |
| **Source** | Utilisateur |
| **Stimulus** | Événement de type sporadic (on ne sait pas précisément quand l'événement aura lieu) |
| **Artéfact** | Serveur de traitement de transactions |
| **Environnement** | Overload |
| **Réponse** | Queue messages |
| **Mesure de la réponse** | En moins de 1 ms |
| **Questions** | 1. Est-ce que certaines transactions peuvent être plus importantes que d’autres? 2. Y a-t-il des moyens pour savoir si le réseau est en surcharge? 3. Combien serait le prix de changer les serveurs pour de l’équipement plus performant? 4. La gestion du trafic peut-elle être optimisée 5. Est-ce que la charge du réseau est plus importante qu’initialement prévu, causant une mauvaise gestion du trafic? |

## Tactique 1 : Increase ressources

**Description**: C’est le fait d'augmenter la puissance de l'infrastructure en place.

**Justification**: Cela permet de d’avoir un système plus performant afin d’avoir des délais plus courts pour la transaction.

## Tactique 2 : Increase resources efficiency

**Description**: C’est le fait d’améliorer les algorithmes utilisés pour augmenter l’efficacité des ressources.

**Justification**: Cela permet d’optimiser le rendement de l’infrastructure et ainsi avoir des réponses plus rapides pour les transactions.

## Tactique 3 : Reduce overhead

**Description**: C’est le fait d’utiliser des intermédiaires pour avoir accès à plus de ressource pour l'événement en cours.

**Justification**: Le fait d’avoir accès à un intermédiaire permet d’augmenter la performance de l’infrastructure temporairement pour ainsi avoir des délais plus courts pour les transactions.

## Vue architecturale (à l’extérieur de la classe)

### Diagramme

### 

### Légende

-Les flèches noires représentent les interactions entre les divers éléments (la nature de l’interaction est spécifié sur la flèche)

-Les carrées blanc représente des éléments physique (équipement de serveur, machine roulant l’application)

-Le carré rouge représente une transaction en cours

-Les flèches blanches représentent les tactiques utilisées

### Texte de description du diagramme

Une transaction est envoyée sur le réseau gorgé de requête vers le serveur. Le serveur 1 reçoit la requête, et remarque que plusieurs autres requête arrive en même temps. Il fait alors appelle au deuxième serveur, qui peut traiter des requêtes en parallèles.

### Table de description des éléments du diagramme

| **Tactique/Élément** | **Description** |
| --- | --- |
| Serveur | Le serveur 1 de l’application |
| Serveur 2 | Les serveurs deux de l’application, afin de diviser les ressources et permettre une meilleure performance lors de la gestion des transactions |
| Achat/Vente | Une transaction en cours |
| Application de bourse | Une instance de l’application de la bourse |
| Reduce overhead | La tactique de reduce overhead, appliqué sur le Serveur |
| Increase ressources | La tactique d’ajout de ressource |
| Increase resources efficiency | La tactique d’am.lioration de la performance des ressources courantes (optimisation d’algorithme) |

### Texte décrivant la relation entre les éléments et les tactiques

Les tactiques Increase ressources et Increase resources efficiency sont des tactiques de prévention. En effet, Increase resources est le fait d’avoir de l’équipement plus optimal et Increase resources efficiency est le fait d’avoir optimisé les algorithmes afin de traiter plus rapidement les requêtes. Seul le reduce overhead est une tactiques qui va être utilisée lors de l’exécution de requête, en temps réel : le serveur 1, lors de la réception de requêtes trop nombreuse, va faire appel à un deuxième serveur afin de traiter des requêtes en parallèle.

# Scénario : Une grande quantité de requête à l’externe via FIX et SAIL dans une très petites période augmentation ainsi la performance nécessaire du système.

| **objectifs d'affaires** | Avoir un temps de réponse en-dessous de 1 ms pour 98% du temps |
| --- | --- |
| **Source** | Interne : Systèmes et moyen de communication |
| **Stimulus** | Requêtes de manières sporadique (intervalles irréguliers) |
| **Artéfact** | L’application (systèmes FIX et SAIL), le serveur |
| **Environnement** | En mode d’opération normale |
| **Réponse** | Assurance des temps des requêtes lors de l’utilisation soit 1ms pour 98% du temps |
| **Mesure de la réponse** | Mesure de la latence des requêtes (ms), Mesure de l’exécution de tâche dans le temps (ms), dénature de la requête (est-ce que la requête a été trop lentement que cela a changé le résultat) |
| **Questions** | 1. Est-ce possible d’avoir les performances désirés ? 2. Lors du 2% non assuré, est-ce que le temps d’attente va quand même être raisonnable? 3. Qu’elles-sont la taille des requêtes? 4. Quelle est le maximum de requêtes reçues à la fois? 5. Quelles sont les spécifications du système utilisé? 6. Quelle est l’importance de chacun des paquets? |

## Tactique 1 : Priorisation des demandes

**Description**: Chaque demande est triée suite à son importance de rapidité d’exécution.

**Justification**: Trier les demandes permet de s’assurer que les transactions envoyés soient gérées immédiatement avec la plus grande performance possible tandis que les requêtes permettant simplement de voir à combien une action est rendue peut être considérée moins importante. Ainsi, la requête sera traitée avec moins d’importance car elle n’affecte pas l’immédiat et vaut mieux que ça soit ces demandes là qui soient affectées par le 2% plus lent que demandé.

**Tactique 2 : Augmentation des ressources disponibles**

**Description**: Mettre à jour le système serveur et ses composantes

**Justification**: Le serveur est probablement rendu vieu et désuet. Le mettre à jour en changeant ces composantes ou le changeant complétement pourrait fort probablement augmenter la performance de la gestion des requêtes . De plus, s’assurer que les équipements tel que la switch utilisé, la vitesse d’internet, etc soient bien suffisantes pour la demande. Finalement, si le système est pour être remplacé car cela est moins cher, le vieux peut être utilisé en parallèle.

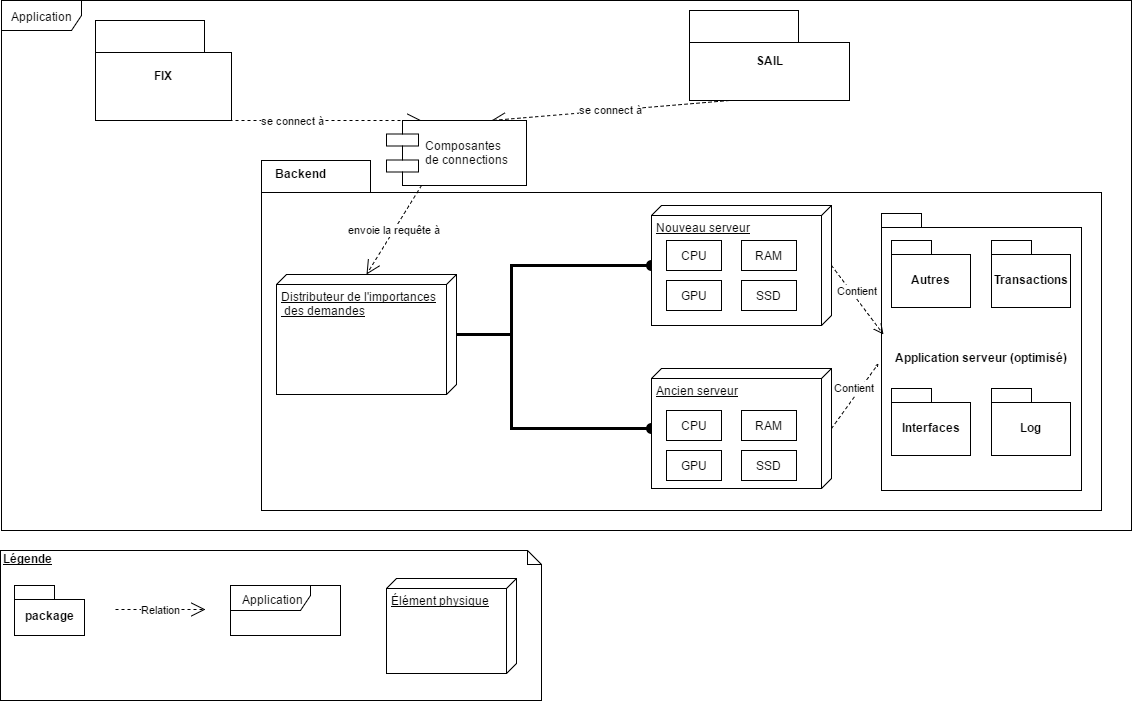
**Tactique 3 : Augmentation de l’efficience des ressources**

**Description**: Refactoring du code source

**Justification**: refactorer le code source de façon stratégique peut augmenter significativement la performance du traitement des requêtes. S’assurer qu’aucune action n’est faite inutilement et que le cpu et la RAM ne soient jamais utilisés inutilement.

**Vue architecturale**

**Diagramme**



### Texte de description du diagramme

Dans ce diagramme, il est possible d’observer les systèmes front-end, soit SAIL et FIX, communiquer avec les composantes de communications. Par la suite, ceux-ci envoient les demandes au distributeur d’importance (moniteur) qui attribue une importance à la demande. Celui-ci envoie les demandes d’après leur importance au nouveau ou au vieux serveur. Finalement, ces serveurs utilise une application optimisée package par package.

### Table de description des éléments du diagramme

| Nom | Description |
| --- | --- |
| SAIL | Package représentent le système SAIL |
| FIX | Package représentent le système FIX |
| Backend | Package représentent le backend du système |
| Composantes de connections | Représente toute les éléments utilisé dans la connection du front et du backend. Donc tout les intermédiaire tel que les switch routeur wi-fi, etc. |
| Distributeur d’importance | Attributeur d’importance à chacune des requêtes |
| Nouveau serveur | Nouveau serveur avec des composantes plus moderne et rapide. |
| Ancien serveur | Ancien serveur antérieurement en place avec les vieux composants |
| Autres | Package contenant toutes les autres packages non-énumérés et optimisés |
| Transaction | Package gérant les différentes transactions et étant optimisé |
| Log | Package comprenant les composants de la journalisation du système |
| Interfaces | Package contenant les différentes interfaces nécessaires au systèmes et ayant été optimisées |

### Texte décrivant la relation entre les éléments et les tactiques

Pour la première tactique qui est la priorisation des demandes, un distributeur de priorité (moniteur) de tâche a été introduit. Ce distributeur reçoit chaque requête et les codifient par importance et longueur de traitement avant de rediriger le packet vers la bonne destination avec la bonne priorité d'exécution. Ceci permet la performance sur chacune des transactions sa gestion dans le temps tandis que la consultation d'une action par exemple, n’est pas affecté si elle n’est pas traité immédiatement. Cela n’affecte donc pas la performance de la requête.

Par la suite, tel qu’identifié sur le diagramme, les demandes sont envoyé sur deux différents serveurs. Les demandes les plus urgentes sont envoyées au nouveau serveur tandis que celles qui le sont moins,

à l’ancien. Cela est en lien direct avec l’augmentation des ressources disponibles. L’augmentation d’un serveur supplémentaire en plus d’utiliser intelligemment les ceux-ci permets d’augmenter la rapidité d’exécution de chacunes des demandes. Le nouveau serveur sera notamment plus performant et traitera les demandes les plus lourdes ou prioritaires tandis que l’ancien traitera des demandes les moins prioritaires ou petites. De plus, la mise à jour des composantes de connections permet d’augmenter la vitesse maximale possible à laquelle le Backend communique avec le Frontend augmentant ainsi sa performance.

Finalement, afin d’augmenter l’efficience des ressources déjà en place, un refactoring de code a été faite dans l’application. L’élimination de cycle, l’utilisation d’interfaces, l’utilisation d’une architecture en couche, l’élimination de zombie utilisant inutilement les ressources sont tous des méthodes de refactoring utilisés. Le diagramme démontre que l’application est propre et bien packagé.