Optimiseur de performance

GUID09058608

Guilardi, Demetrio

UQAC
Université du Québec
à Chicoutimi

555 Boulevard de l'Université
Chicoutimi, QC G7H 2B1
Département d'informatique et de mathématique
Cours 8INF851 - Génie logiciel

Table des Matières

1. Introduction

Ce travail présente un rapport pour une solution donnée pour un problème de performance logiciel. Ce travail commence par un scénario de performance bien défini, que je présenterai dans la section suivante. Je propose également une solution pour résoudre ce problème, en ciblant les deux problèmes de performance identifiés: *(i)* le **déséquilibre** et *(ii)* la **surcharge**.

Le reste de ce rapport est organisé comme suit: Dans la section 2, je décris le problème et le scénario; Dans la section 3, je présente ma solution, son architecture, son fonctionnement et quelques exemples de sortie; Dans la section 4, je présente les résultats de deux adaptateurs différents, standard et optimisé; Toujours dans la section 4, je compare les deux résultats. Dans la section 5, je fais quelques discussions sur les avantages, les risques, les couches fonctionnelles et la stratégie de sécurité; Dans la section 6, je conclus ce travail, ainsi qu'un éventuel développement futur à grande échelle.

2. La Problématique

Les techniques de performance et d'optimisation sont des sujets bien discutés dans le domaine du génie logiciel. Woodside et al., 2007, ont conclu que l'ingénierie des performances logicielles est faible en matière de prédiction, de test et de mesure. Des frameworks axés sur la performance logicielle sont également disponibles, tels que *Kieker*, Van Hoorn et al., 2012, ont développé le cadre *Kieker*, un frameworks extensible pour la surveillance et l'analyse du comportement d'exécution des systèmes concurrents et distribués.

Dans ce travail, un problème de performance est présenté. Un système effectue le traitement de requête provenant de divers fournisseurs. Chaque fournisseur effectue ses envois de manière asynchrone et chaque requête est insérée et entreposée dans un transformateur.

Chaque transformateur dispose d'un *buffer* capable de mettre en file d'attente et de mettre en *cache* un certain nombre de requêtes. Le nombre de requêtes en file d'attente détermine la charge d'un transformateur. Plus un transformateur est chargé, plus les coûts de traitement des demandes sont élevés. Les coûts associés au traitement de chaque demande ne sont pas liés au traitement ou au temps de fonctionnement, bien que directement liés à la charge du transformateur.

Dans ce scénario, les fournisseurs débordent les transformateurs, provoquant les problèmes de performances suivants:

(i) **Déséquilibre**. Les transformateurs ne sont pas également équilibrés, certains peuvent avoir une charge considérablement plus élevée, tandis que d'autres peuvent être sous-chargés.

(ii) **Surcharge**. À mesure que les transformateurs reçoivent plus de demandes, ces demandes sont mises en file d'attente et les charges des transformateurs sont augmentées. Les transformateurs surchargés dépensent plus pour traiter les demandes, ce qui augmente les coûts des demandes.

Dans la section suivante, je présente ma solution à ces problèmes de performances.

3. La Solution

Ma solution est un système Java, composé de deux adaptateurs qui implémentent à la fois une solution déséquilibrée et surchargée à faible performance, ainsi que la solution optimisée. La mise en œuvre des deux solutions permet d'effectuer des analyses et des comparaisons sur les données de résultats collectées à partir des deux adaptateurs.

3.1. Architecture

La **figure 1** montre la vue d'ensemble de l'architecture. La case grise du haut représente les fournisseurs. Chaque fournisseur a un mécanisme de bouclage qui fait une demande toutes les microsecondes `REQUEST_INTERVAL`, qui est défini dans `Config.class`. Le système reçoit chaque demande, vérifie la charge des transformateurs, décide de recevoir la demande ou de la rejeter. Les demandes acceptées sont mises en cache dans le *buffer* du transformateur associé au fournisseur demandeur.

Toujours dans la **figure 1**, le système représenté fonctionne en boucle. Dans chaque cycle, le système: *(i)* Exécute la méthode de pré-boucle dans l'adaptateur; *(ii)* Vérifier le solde; *(iii)* boucle dans chaque transformateur, appelant la méthode associée à l'adaptateur; *(iv)* exécute une méthode de post-boucle dans l'adaptateur. Les transformateurs traitent les demandes lors de l'étape *(iii) (boucle dans chaque transformateur)*. La réponse est renvoyée au fournisseur juste après son traitement, via l'instance système.

Les demandes sont rejetées par le système au moment où elles sont reçues, si la vérification de charge échoue. Dans le cas où les demandes sont acceptées, elles sont mises en mémoire *buffer* dans une file d'attente du transformateur associé au fournisseur. Les demandes acceptées sont traitées par le système, après avoir été traitées par le transformateur.

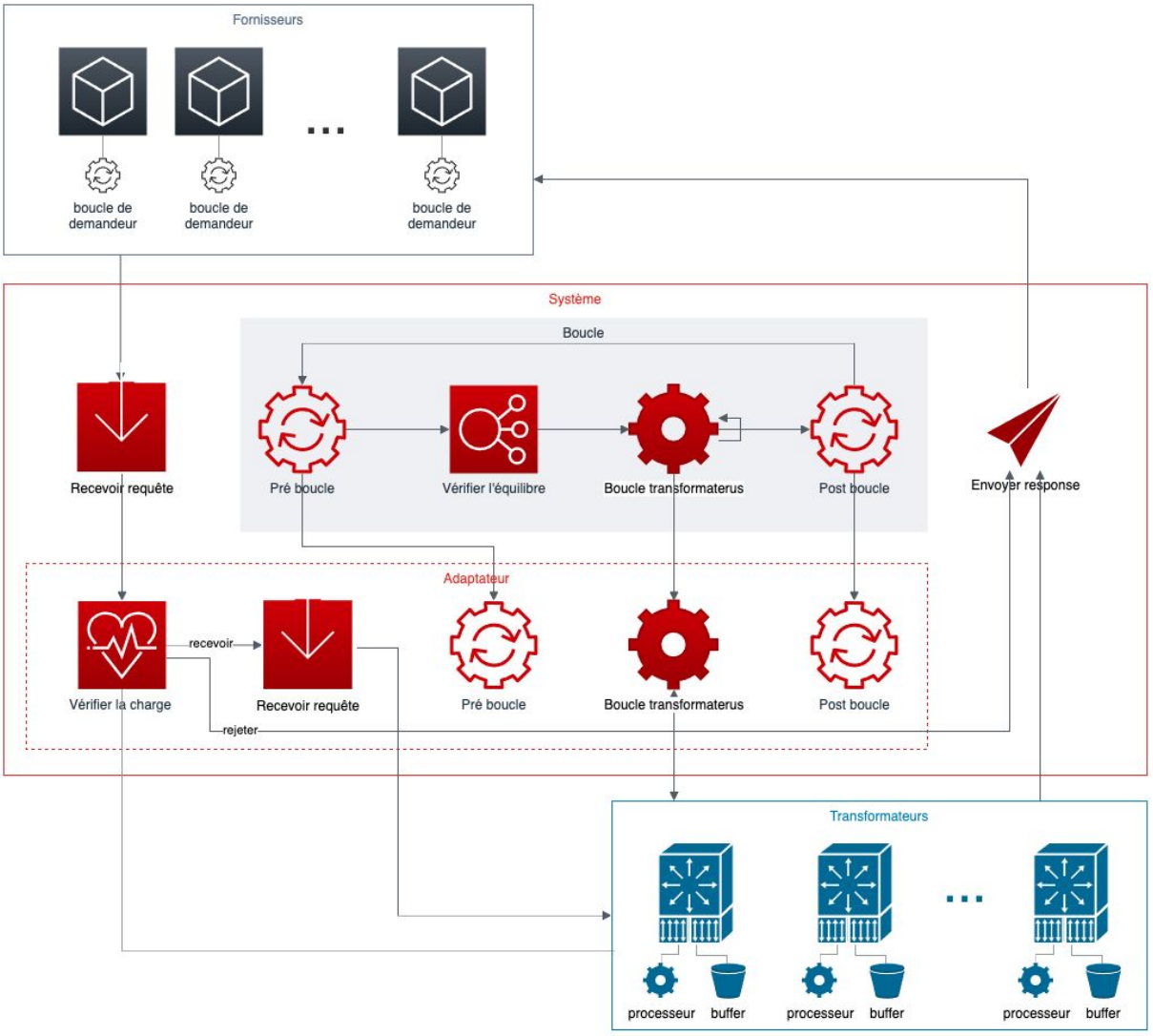

Figure 1. Présentation de l'architecture

Les composants internes du système, les adaptateurs et les interfaces sont représentés dans le diagramme de classes de la **figure 2**. Le système est initialisé avec le nom de classe complet d'un adaptateur. Le système exécute une technique de réflexion pour récupérer le *bytecode* de l'adaptateur et s'initialiser. La technique de réflexion donne lieu à des adaptateurs plus flexibles et plus adaptatifs.

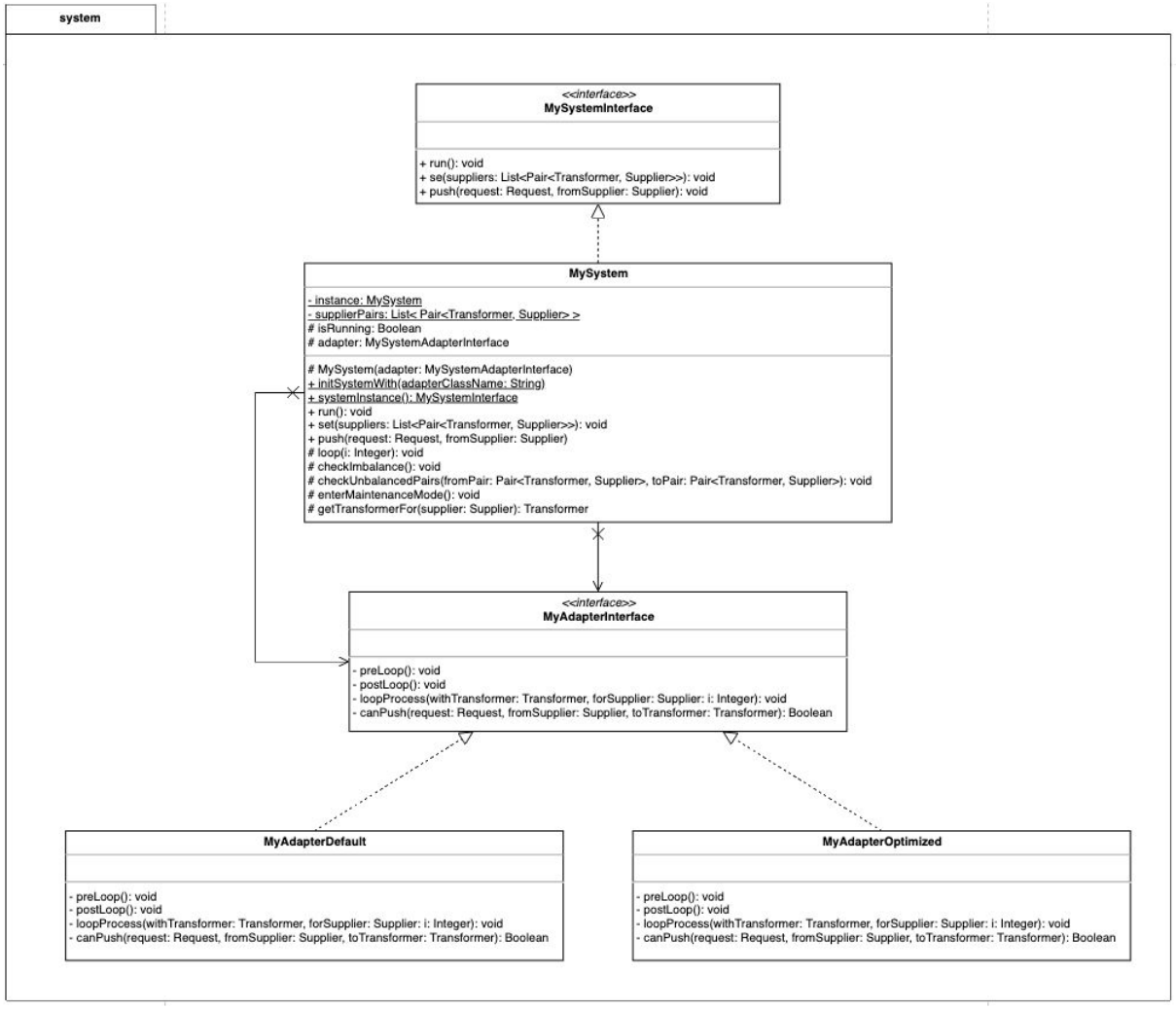

Figure 2. Diagramme de classes du système

3.2. Étalonnage

Dans ma solution, le système est calibré en modifiant les paramètres dans un fichier de classe nommé `Config.class`. Dans cette section, je décris chacun des paramètres, ainsi que leurs effets d'entraînement lorsqu'ils sont modifiés. **Listage 1** contient toutes les variables de `Config.class` liées à l'étalonnage. Les valeurs de la liste sont également celles par défaut.

```
public static final Integer NUM_SUPPLIERS = 4;
public static final Integer LOOP_INTERVAL = 2;
public static final Integer NUM_LOOPS_TO_MAINTENANCE = 5000;
public static final Integer IMBALANCE_THRESHOLD = 2;
public static final Integer REQUEST_INTERVAL = 3;
public static final Integer REQUEST_PROCESS_TIME_MIN_MS = 1;
public static final Integer REQUEST_PROCESS_TIME_MAX_MS = 3;
public static final Integer TRANSFORMER_BUFFER_OK = 3;
public static final Integer TRANSFORMER_BUFFER_DANGER = 7;
public static final Integer TRANSFORMER_BUFFER_MAX = 10;
```

Listage 1. Exemple de configuration système

`NUM_SUPPLIERS`: Définit la quantité de fournisseurs et de transformateurs, car chaque fournisseur est lié à un transformateur.

`LOOP_INTERVAL`: Définit l'intervalle de la boucle principale du système, en microsecondes. Plus la valeur est basse, plus le système interagira rapidement avec ses boucles, plus les appels aux adaptateurs et aux transformateurs seront rapides. Le temps de traitement général des requêtes (en transformateurs) n'est pas affecté par ce paramètre.

`NUM_LOOPS_TO_MAINTENANCE`: Définit la quantité totale de boucles que prend le système avant de passer en mode maintenance et affiche les résultats. Le temps total d'exécution est directement affecté par ce paramètre. *executionTime = NUM_LOOPS_TO_MAINTENANCE * LOOP_INTERVAL*.

`IMBALANCE_THRESHOLD`: Définit quelle est la différence qui définit si les transformateurs sont déséquilibrés. Les transformateurs avec une différence de charge dans ce paramètre seront considérés comme ok. Si une différence de charge dépasse la valeur de ce paramètre, le transformateur avec la charge la plus élevée est considéré comme déséquilibré et ses demandes voient leurs coûts augmentés.

`REQUEST_INTERVAL`: Définit l'intervalle de temps, en microsecondes, de la boucle de demande des fournisseurs. Plus la valeur est basse, plus les fournisseurs envoient rapidement de nouvelles demandes au système.

`REQUEST_PROCESS_TIME_MIN_MS` et `REQUEST_PROCESS_TIME_MAX_MS`: Ces deux paramètres définissent la plage de temps de traitement de chaque demande.

`TRANSFORMER_BUFFER_OK` et `TRANSFORMER_BUFFER_DANGER`: Ces deux paramètres définissent les valeurs qui définissent quand les charges des transformateurs sont considérées comme *ok* ou en *danger*. Ce sont des références au nombre de requêtes en file d'attente dans les *buffers* du transformateur.

`TRANSFORMER_BUFFER_MAX`: Définit le nombre maximal de requêtes pouvant être mises en file d'attente dans les *buffers* du transformateur.

3.3. Fonctionnement

The operating mechanism is a five-steps straight forward process: (i) The system is initialized with an adapter's full class name (see section 3, subsection 3.1); (ii) The suppliers are constructed; (iii) The system is set with the suppliers list; (iv) The system is started; (v) The suppliers are started.

Le mécanisme du système est un processus simple en cinq étapes: *(i)* Le système est initialisé avec le nom de classe complet d'un adaptateur (voir section 3, sous-section 3.1); *(ii)* Les fournisseurs sont construits; *(iii)* Le système est défini avec la liste des fournisseurs; *(iv)* Le système est démarré; *(v)* Les fournisseurs sont lancés.

Listage 2 affiche un exemple de méthode `main`, contenant les cinq étapes nécessaires pour exécuter le système.

```
public static void main(String[] args){
    initSystemWith("ca.uqac.system.MyAdapterDefault");
    initSupplierPairs();
    systemInstance().set(supplierPairs);
    systemInstance().run();
    startSuppliers();
}
```

Listage 2. Example of main method `main`.

Pour aider les chercheurs à mieux analyser et suivre la progression du système, une interface utilisateur auxiliaire peut également être utilisée. Cet auxiliaire surveille toute liste donnée de paires fournisseurs-transformateurs. L'interface utilisateur peut être démarrée à tout moment, avec la liste des paires fournisseurs-transformateurs comme paramètre. L'interface utilisateur est initialisée en appelant simplement la méthode `Gui.initGui(supplierPairs)`.

Pendant le fonctionnement du système, si l'interface utilisateur auxiliaire est initialisée, il est lancé une interface utilisateur simple, contenant des barres indiquant la charge actuelle de chaque transformateur.

La **figure 3** contient un exemple de l'interface utilisateur auxiliaire. Chaque barre représente la charge d'un transformateur. Les textes au-dessus des barres indiquent chaque état possible, *idle*, *ok*, *danger* et *max*.

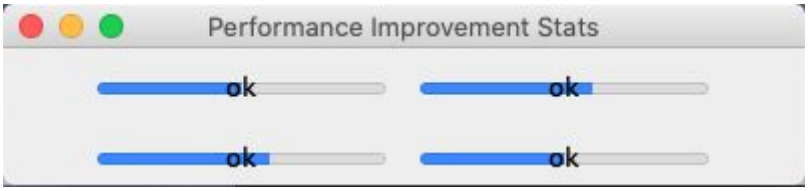

Figure 3. L'interface utilisateur auxiliaire.

3.4. Sortie

Une fois que le système a exécuté toutes ses boucles et est entré en mode maintenance (voir la section 3.2), le système produit un rapport contenant un résumé pour chaque transformateur, ainsi que les résultats du système, avec les totaux et les moyennes.

La **figure 4** contient un exemple de rapport de sortie. Les deux premières données tabulaires contiennent des données résumées représentant deux fournisseurs (n° 2 et n° 3). La première colonne décrit les données, la deuxième colonne affiche les quantités, la troisième colonne le coût de traitement. Les dernières données tabulaires contiennent le récapitulatif du système, avec le nombre de demandes, le succès, le rejet, les demandes perdues et les moyennes des coûts, ainsi que le coût total.

```
==== SUMMARY OF EVENTS FOR SUPPLIER #2 ====
|     type      |    qtd.    |    cost    |
| ------------- | ---------- | ---------- |
| default cost  |       3730 |       3730 |
| imbalan. cost |         87 |        174 |
| overload cost |         35 |        105 |
| rejected      |         50 |       1250 |
| lost          |          5 |      n/a   |
| total         |       3785 |       5259 |
| ------------- | ---------- | ---------- |

==== SUMMARY OF EVENTS FOR SUPPLIER #3 ====
|     type      |    qtd.    |    cost    |
| ------------- | ---------- | ---------- |
| default cost  |       3632 |       3632 |
| imbalan. cost |         66 |        132 |
| overload cost |          8 |         24 |
| rejected      |        149 |       3725 |
| lost          |          5 |      n/a   |
| total         |       3786 |       7513 |
| ------------- | ---------- | ---------- |

============ MY SYSTEM RESULTS ============
request avg  :       3787
success avg  :       3728
rejected avg :         54
lost avg     :          5
cost avg     :       5375
cost total   :      21500
```

Figure 4. Exemple de sortie.

4. Résultats

Dans cette section, je présente les résultats des deux adaptateurs, le standard et l'optimisé. Après avoir présenté, je compare également les résultats.

4.1. Les résultats de l'adaptateur standard

Comme indiqué dans la section 3, sous-section 3.2, j'ai défini la propriété `SYSTEM_ADAPTER` avec `"ca.uqac.performance.system.MyAdapterDefault"`. J'ai ensuite exécuté le système cinq fois, collecté et consolidé toutes les sorties dans une seule table contenant les sommes, les minimums, les maximums et les moyennes.

Le tableau 1 présente les résultats des cinq tests. Il contient les quantités de réponses à la demande, les coûts totaux, les coûts minimums, les coûts maximums et les coûts moyens. Les coûts minimum, maximum et moyen sont calculés en divisant par le nombre de transformateurs.

Le transformateur le plus efficace a traité ses demandes avec un coût total de 9703, lors du test 2 (cellule D17 du tableau 1). Alors que le transformateur le moins efficace du même test, a traité ses demandes avec un coût total de 16251, dans le test 2 (cellule E17 du tableau 1). De plus, le transformateur le moins efficace de tous les tests a traité ses demandes avec un coût total de 16843, dans le test 1 (cellule E9 du tableau 1). La proportion entre les transformateurs plus efficaces et les moins efficaces est aussi basse que 0,60, pour le même test (test 2), comme le montrent les résultats du test 2 (cellule G17). **Les résultats indiquent que cet adaptateur a le problème de déséquilibre**.

Les coûts de déséquilibre et de surcharge sont responsables de 60,4% du coût total dans le meilleur des cas (test 2), ces mêmes indicateurs sont responsables de 62,3% du coût total dans le pire des cas (test 3). De plus, les coûts de surcharge sont 2 à 3 fois plus élevés que les coûts de déséquilibre. **Ces résultats indiquent que cet adaptateur a également le problème de surcharge**.

	A	B	C	D	E	F	G
1	**MyAdapterDefault**						
2	**Test #1**						
3	**responses**	**qtd.**	**cost (tot)**	**cost (min)**	**cost (max)**	**cost (avg)**	**cost (min/max)**
4	success	14873	14873	3700	3736	3718.25	0.99
5	imbalanced	4594	9188	1614	2892	2297	0.56
6	overloaded	10015	30045	6846	8103	7511.25	0.84
7	rejected	356	8900	1875	2650	2225	0.71
8	lost	37	n/a	n/a	n/a	n/a	n/a
9	total	15266	63006	14622	16843	15751.5	0.87
10	**Test #3**						
11	**responses**	**qtd.**	**cost (tot)**	**cost (min)**	**cost (max)**	**cost (avg)**	**cost (min/max)**
12	success	14869	14869	3683	3753	3717.25	0.98
13	imbalanced	5134	10268	1714	3446	2567	0.50
14	overloaded	7009	21027	3636	6822	5256.75	0.53
15	rejected	223	5575	600	2300	1393.75	0.26
16	lost	32	n/a	n/a	n/a	n/a	n/a
17	total	15124	51739	9703	16251	12934.75	0.60
18	**Test #3**						
19	**responses**	**qtd.**	**cost (tot)**	**cost (min)**	**cost (max)**	**cost (avg)**	**cost (min/max)**
20	success	14923	14923	3716	3760	3730.75	0.99
21	imbalanced	5277	10554	2142	2892	2638.5	0.74
22	overloaded	8194	24582	5547	7080	6145.5	0.78
23	rejected	253	6325	950	1950	1581.25	0.49
24	lost	35	n/a	n/a	n/a	n/a	n/a
25	total	15211	56384	12399	15638	14096	0.79
26	**Test #4**						
27	**responses**	**qtd.**	**cost (tot)**	**cost (min)**	**cost (max)**	**cost (avg)**	**cost (min/max)**
28	success	14884	14884	3704	3735	3721	0.99
29	imbalanced	5013	10026	1454	3392	2506.5	0.43
30	overloaded	7632	22896	5076	6936	5724	0.73
31	rejected	233	5825	1025	1875	1456.25	0.55
32	lost	32	n/a	n/a	n/a	n/a	n/a
33	total	15149	53631	11440	15907	13407.75	0.72
34	**Test #5**						
35	**responses**	**qtd.**	**cost (tot)**	**cost (min)**	**cost (max)**	**cost (avg)**	**cost (min/max)**
36	success	14865	14865	3704	3725	3716.25	0.99
37	imbalanced	4746	9492	2130	2766	2373	0.77
38	overloaded	9064	27192	5610	7317	6798	0.00
39	rejected	296	7400	1625	2225	1850	0.73
40	lost	36	n/a	n/a	n/a	n/a	n/a
41	total	15198	58974	13158	15514	14743.5	0.85

Tableau 1. Résultats des cinq tests avec `MyAdapterDefault`.

4.2. Les résultats de l'adaptateur optimisé

Comme indiqué dans la section 3, sous-section 3.2, j'ai défini la propriété `SYSTEM_ADAPTER` avec `"ca.uqac.performance.system.MyAdapterOptimized"`. J'ai ensuite exécuté le système cinq fois, collecté et consolidé toutes les sorties dans une seule table contenant les sommes, les minimums, les maximums et les moyennes.

Le tableau 2 présente les résultats des cinq tests. Il contient les quantités de réponses à la demande, les coûts totaux, les coûts minimums, les coûts maximums et les coûts moyens. Les coûts minimum, maximum et moyen sont calculés en divisant par le nombre de transformateurs. Le transformateur le plus efficace a traité ses demandes avec un coût total de 5210, dans le test 2 (cellule D17 du tableau 2). Alors que le transformateur le moins efficace du même test, a traité ses demandes avec un coût total de 5810, lors du test 2 (cellule E17 du tableau 2). De plus, le transformateur le moins efficace parmi tous les tests a traité ses demandes avec un coût total de 6157, dans le test 3 (cellule E25 dans le tableau 2).

La proportion entre les transformateurs plus efficaces et les moins efficaces est aussi basse que 0,90, pour le même test (test 2), comme le montrent les résultats du test 2 (cellule G17). **Les résultats indiquent que cet adaptateur ne présente pas le problème de déséquilibre.**

Les résultats montrent également qu'il n'y a aucun coût lié à la surcharge. Aucun des transformateurs n'avait traité une seule demande en état de surcharge. **Ces résultats indiquent que cet adaptateur n'a pas le problème de surcharge**.

	A	B	C	D	E	F	G
1	**MyAdapterOptimized**						
2	**Test #1**						
3	**responses**	**qtd.**	**cost (tot)**	**cost (min)**	**cost (max)**	**cost (avg)**	**cost (min/max)**
4	success	14918	14918	3722	3741	3729.5	0.99
5	imbalanced	236	472	76	134	118	0.57
6	overloaded	0	0	0	0	0	0.00
7	rejected	265	6625	1400	1825	1656.25	0.77
8	lost	20	n/a	n/a	n/a	n/a	n/a
9	total	15203	22015	5198	5685	5503.75	0.91
10	**Test #2**						
11	**responses**	**qtd.**	**cost (tot)**	**cost (min)**	**cost (max)**	**cost (avg)**	**cost (min/max)**
12	success	14868	14868	3675	3750	3717	0.98
13	imbalanced	314	628	142	170	157	0.84
14	overloaded	0	0	0	0	0	0.00
15	rejected	257	6425	1325	1975	1606.25	0.67
16	lost	27	n/a	n/a	n/a	n/a	n/a
17	total	15152	21921	5210	5810	5480.25	0.90
18	**Test #3**						
19	**responses**	**qtd.**	**cost (tot)**	**cost (min)**	**cost (max)**	**cost (avg)**	**cost (min/max)**
20	success	14856	14856	3685	3735	3714	0.99
21	imbalanced	307	614	128	180	153.5	0.71
22	overloaded	0	0	0	0	0	0.00
23	rejected	325	8125	1875	2250	2031.25	0.83
24	lost	27	n/a	n/a	n/a	n/a	n/a
25	total	15209	23595	5720	6157	5898.75	0.93
26	**Test #4**						
27	**responses**	**qtd.**	**cost (tot)**	**cost (min)**	**cost (max)**	**cost (avg)**	**cost (min/max)**
28	success	14930	14930	3694	3769	3732.5	0.98
29	imbalanced	292	584	124	180	146	0.69
30	overloaded	0	0	0	0	0	0.00
31	rejected	264	6600	1325	1925	1650	0.69
32	lost	28	n/a	n/a	n/a	n/a	n/a
33	total	15223	22138	5212	5799	5534.5	0.90
34	**Test #5**						
35	**responses**	**qtd.**	**cost (tot)**	**cost (min)**	**cost (max)**	**cost (avg)**	**cost (min/max)**
36	success	14899	14899	3714	3734	3724.75	0.99
37	imbalanced	277	554	128	158	138.5	0.81
38	overloaded	0	0	0	0	0	0.00
39	rejected	292	7300	1675	2025	1825	0.83
40	lost	25	n/a	n/a	n/a	n/a	n/a
41	total	15216	22753	5535	5902	5688.25	0.94

Tableau 2. Résultats des cinq tests avec `MyAdapterOptimized`.

4.3. Comparaison des résultats

Après avoir collecté 5 résultats de chaque adaptateur (par défaut et optimisé), j'ai fait un tableau contenant les coûts totaux (tous les adaptateurs) de chaque test, répartis en coûts de réussite, de déséquilibre, de surcharge et de rejet.

La figure 5 contient les données de comparaison et le graphique pour les deux adaptateurs. Ces données affichent clairement les principales différences des adaptateurs. Alors que les tests liés à l'adaptateur par défaut ont des coûts de déséquilibre et de surcharge élevés, les tests liés à l'adaptateur optimisé ont peu de coûts de déséquilibre et aucun coût de surcharge.

Cette comparaison des coûts montre à quel point l'adaptateur optimisé est plus efficace que l'adaptateur par défaut.

	MyAdapterDefault					MyAdapterOptimized				
TOTAL COSTS	**default #1**	**default #2**	**default #3**	**default #4**	**default #5**	**opt. #1**	**opt. #2**	**opt. #3**	**opt. #4**	**opt. #5**
success	14873	14869	14923	14884	14865	14918	14868	14856	14930	14899
imbalance	9188	10268	10554	10026	9492	472	628	614	584	554
overload	30045	21027	24582	22896	27192	0	0	0	0	0
rejection	8900	5575	6325	5825	7400	6625	6425	8125	6600	7300

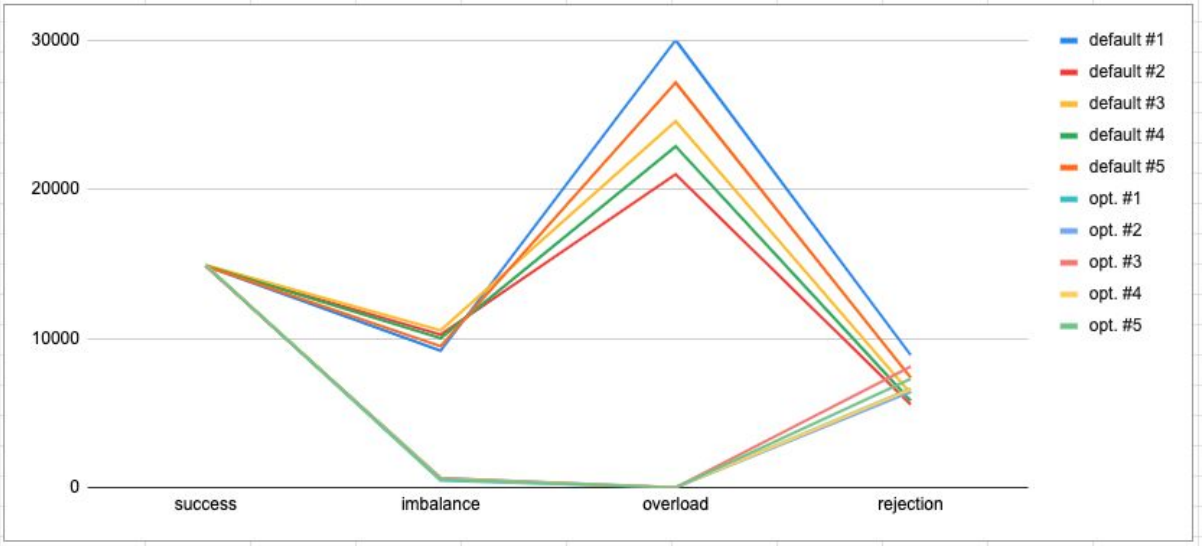

Figure 5. Tableau de données et de comparaison.

5. Discussions

Dans cette section, je fais quelques discussions sur les avantages et les inconvénients, ainsi que sur les risques et l'efficacité; Dans cette section, je fais quelques réflexions sur les couches fonctionnelles et la stratégie de sécurité.

4.1. Avantages

Dans mon approche, les adaptateurs sont connectés au système, offrant la flexibilité nécessaire pour s'adapter à différents scénarios d'utilisation. On peut développer des adaptateurs adaptatifs, artificiellement intelligents ou tout autre type d'adaptateur pour répondre à différents objectifs.

De plus, ma solution utilise la technique de réflexion pour créer des adaptateurs. Cette technique permet au système «d'utiliser des classes externes définies par l'utilisateur en créant des instances d'objets d'extensibilité en utilisant leurs noms complets» (*Trail: The Reflection API*, n.d.).

4.2. Risques

Le risque majeur identifié lors de l'élaboration de cette étude est que le coût des ressources informatiques de l'optimisation est supérieur au coût de traitement des demandes. Lorsque le volume de requêtes ne dépasse pas suffisamment pour déborder du système, les algorithmes d'optimisation peuvent coûter plus de ressources que le traitement des requêtes elles-mêmes.

Un autre risque observé lors du développement de cette étude est que les adaptateurs fonctionnent différemment selon l'étalonnage. Ce risque consiste à trouver la bonne correspondance d'algorithme et d'étalonnage pour obtenir de meilleurs résultats.

4.4. Couches fonctionnelles et stratégie de sécurité

Dans mon approche, le système fonctionne comme un orchestrateur. Il effectue une boucle en étapes prédéterminées, appelant les méthodes d'un adaptateur accroché. La simplicité, bien que des étapes bien définies, permettent de développer des adaptateurs de manière standardisée.

Comme le montre la figure 1, les adaptateurs interfaçant également les transformateurs avec les fournisseurs. La couche d'adaptateurs **protège le système contre les appels directs des adaptateurs ou des fournisseurs**, le gardant à l'abri des interférences, fournissant un noyau plus fiable.

6. Conclusion

Dans cette étude, il a introduit un problème de performance, dans lequel un système reçoit et traite des demandes de fournisseurs qui **déséquilibrent** et **surchargent** sûrement le système. J'ai présenté une solution basée sur la construction d'un système qui traite les demandes via des adaptateurs, en utilisant la technique de réflexion. J'ai développé le système avec deux adaptateurs: (i) **standard** et (ii) **optimisé**.

Après avoir développé le système et les deux adaptateurs, j'ai effectué 10 tests et collecté les résultats de manière égale, 5 à partir de la norme et 5 à partir de l'adaptateur optimisé. Avec les résultats des deux adaptateurs, j'ai également fait des comparaisons qui ont conduit à des conclusions.

Travail futur

Les travaux futurs pourraient se concentrer sur la construction de plus d'adaptateurs. Le système lui-même pourrait également être optimisé pour changer dynamiquement les adaptateurs. De plus, les travaux futurs pourraient se concentrer sur l'atténuation des risques. Une optimisation pourrait identifier des déséquilibres au détriment de l'équilibrage lui-même, lorsque le coût de l'équilibrage dépasse le coût du traitement des demandes. Enfin, des travaux futurs pourraient calculer l'étalonnage du système, de manière statique ou dynamique.

Références

Trail: The Reflection API. (n.d.). ORACLE Java Documentation. Retrieved November 18, 2020, from https://docs.oracle.com/javase/tutorial/reflect/index.html

van Hoorn, A., Waller, J., & Hasselbring, W. (2012, April). Kieker: a framework for application performance monitoring and dynamic software analysis. *ACM/SPEC International Conference on Performance Engineering*. ACM. 10.1145/2188286.2188326

Woodside, M., Franks, G., & Petriu, D. C. (2007). The Future of Software Performance Engineering. *Future of Software Engineering (FOSE '07)*, *2007*. IEEE. 10.1109/FOSE.2007.32