MINISTERE DE L'ENSEIGNEMENT SUPERIEUR ET DE LA RECHERCHE SCIENTIFIQUE

UNIVERSITE ABDELHAMID IBN BADIS - MOSTAGANEM

Faculté des Sciences Exactes et d'Informatique

Département de Mathématiques et informatique

Filière : Informatique

MEMOIRE DE FIN D'ETUDES

Pour l'Obtention du Diplôme de Master en Informatique

Option : **Ingénierie des Systèmes d'Information**

Présenté par :

FEKIRA Meriem et BOUZIANE Hichem

THEME :

Utilisation des votes pondérés dans la gestion de la cohérence des données dans les systèmes distribués à grande échelle

Soutenu le : 19-06-2021

Devant le jury composé de :

M. BETOUATI F	Université de Mostaganem	Président
M. MEGHOUFEL B	Université de Mostaganem	Examinateur
M. BOUHARAOUA Farouk	Université de Mostaganem	Encadreur

Année Universitaire 2020-2021

Dédicaces

A mon très cher père qui me trouve la source de sa fierté, qui a toujours répondu présent dans les moments les plus difficiles, près de moi pour m'écouter, me soutenir, m'encourager et me donnée force à poursuivre mes études.

A ma chère Maman qui m'a encouragé avec la confiance qui m'a fait tout au long de mes études, qui m'a donné l'aide et l'amour qui m'ont permis d'écrire ce mémoire dans les meilleures conditions.

Aucune dédicace ne pourra compenser les sacrifices de mes parents, qui sans eux ma réussite n'aura pas lieu.

A mes adorables sœurs Fatima zahra et Soumia et sa fille Ratil qui m'a toujours soutenu.

A ma cher frère Mohamed.

A mes amis surtout mon binôme BOUZIANE Hichem en témoignage de nos sincères reconnaissances pour les efforts qu'ils ont consentis pour nous soutenir au cours de nos études. Que dieu nous garde toujours unis.

A toute personne qui aide nous à faire notre mémoire.

FEKIRA Meriem

Dédicaces

A mon très cher père qui me trouve la source de sa fierté, qui a toujours répondu présent dans les moments les plus difficiles, près de moi pour m'écouter, me soutenir, m'encourager et me donnée force à poursuivre mes études.

A ma chère Maman qui m'a encouragé avec la confiance qui m'a fait tout au long de mes études, qui m'a donné l'aide et l'amour qui m'ont permis d'écrire ce mémoire dans les meilleures conditions.

Aucune dédicace ne pourra compenser les sacrifices de mes parents, qui sans eux ma réussite n'aura pas lieu.

A mes adorables sœurs et mes chers frères qui m'a toujours soutenu.

A mes amis surtout mon binôme FEKIRA Meriem en témoignage de nos sincères reconnaissances pour les efforts qu'ils ont consentis pour nous soutenir au cours de nos études. Que dieu nous garde toujours unis.

A toute personne qui aide nous à faire notre mémoire.

BOUZIANE Hichem

Remerciements

Nous tenons à exprimer nos sincères remerciements et toute gratitude à notre encadreur Monsieur BOUHARAOUA Farouk, Enseignant à l'Université Abdelhamid Ibn Badis Mostaganem, pour la confiance qu'il nous a manifestée tout au long de cette année. Malgré ses lourdes charges, il a toujours été là pour nous orienter et nous soutenir.

Nous espérons avoir étiez à la hauteur de ses espérances.

Résumé

La réplication des données est une importante technique utilisée dans les systèmes distribués à grande échelle qui vise à améliorer la disponibilité des données et les performances. Lors de la mise à jour des répliques des données, le problème du maintien de la cohérence entre les répliques doit être engendré.

Plusieurs travaux ont été réalisés dans le domaine de la cohérence des répliques dans les systèmes distribués. Parmi ces travaux, nous nous intéressons à l'approche basée sur les votes pondérés pour les systèmes des quorums.

Mots-clés :

Systèmes distribués, réplication de données, cohérence, quorum, votes.

Abstract

Data replication is an important technique used in large-scale distributed systems that aims to improve data availability and performance.

When updating the data replicas, the problem of maintaining consistency between the replicas must be raised.

Several works have been carried out in the field of the consistency of replicas in distributed systems. Among these works, we are interested in the approach based on weighted votes for quorum systems.

Keywords:

Distributed systems, data replication, consistency, quorum, votes.

ملخص

يعد تكرار البيانات تقنية مهمة تستخدم في الأنظمة الموزعة على نطاق واسع والتي تهدف إلى تحسين توافر البيانات وأدائها.

عند تحديث النسخ المتماثلة للبيانات، يجب أن تظهر مشكلة الحفاظ على التناسق بين النسخ المتماثلة.

تم تنفيذ العديد من الأعمال في مجال تناسق النسخ المتماثلة في الأنظمة الموزعة. من بين هذه الأعمال، نحن مهتمون بالنهج القائم على الأصوات المرجحة لأنظمة النصاب القانوني.

الكلمات الرئيسية:

الأنظمة الموزعة، تكرار البيانات، التناسق، النصاب، أصوات.

Liste des figures

Liste des tableaux

Liste des abréviations

Abréviation	Expression Complète	Page
SDGE	**S**ystem **D**istribué a **G**rande **E**chelle	1
CRUD	**C**reate, **R**ead, **U**pdate, **D**elete	10
CAP	**C**onsistency, **A**vailability, **P**artition tolerance	11
ROWA	**R**ead **O**ne **W**rite **A**ll	12
ROWAA	**R**ead **O**nce **W**rite **A**ll **A**vailable	12
UML	**U**nified **M**odeling **L**anguage	19
CPU	**C**entral **P**rocessing **U**nit	29
IDE	**I**ntegrated **D**evelopment **E**nvironment	29
XML	l'e**X**tensible **M**arkup **L**anguage	30
PHP	**H**ypertext **P**re**P**rocessor	30
HTML	**H**yper**T**ext **M**arkup **L**anguage	30
RAM	**R**andom **A**ccess **M**emory	30
BP	**B**ande **P**assante	30

Table des matières

Introduction Générale

Ces dernières décennies ont été marquées par les systèmes distribués à grande échelle (SDGE) qui ont participé un avancement majeur pour la communauté scientifique.

Le terme « *passage à l'échelle* » est utilisé pour désigner la capacité d'un système à passer d'une distribution faible (peu de membres, membres proches, réseau local) à une distribution plus large (beaucoup de membres, temps de communication important entre les membres).

L'expérimentation en grandeur réelle n'est pas réellement envisageable, car il n'est pas possible de déployer des expériences à grande échelle faute d'accès global. D'autre part, il n'est pas possible non plus de récupérer des données d'utilisation réelle car celles-ci sont toujours dans une phase expérimentale et pas en production.

Les techniques de partage, utilisées par les SDGE, sont le plus souvent basées sur le principe de réplication pour assurer une disponibilité très élevée des données.

Lorsqu'un nœud tombe en panne, une copie des données peut toujours être récupérée à partir d'un autre emplacement où une autre copie est disponible. Dans ce contexte, un même objet (donnée, base de données, fichier, service, programme, etc.) peut être répliqué en autant de copies que nécessaires. Chaque copie étant détenue et accédée par un ou plusieurs utilisateurs.

Bien que très avantageuse, la réplication présente tout de même un inconvénient majeur de cohérence des données où toutes les copies d'une même donnée doivent être théoriquement identiques, mais les techniques de réplication n'assurent pas une cohérence sur l'ensemble des données répliquées. Ainsi, la mise en œuvre des opérations de lecture et d'écriture demeure difficile.

De ce fait, plusieurs approches ont été proposées dans le but de maintenir la cohérence des répliques et garantir une bonne gestion de cette cohérence dans les SDGE.

Parmi ces techniques nous nous intéressons à la technique de vote pondéré qui est basée sur le principe des quorums où chaque nœud doit avoir un poids pour faire le suffrage.

Dans ce contexte et afin de présenter les techniques de gestion de la cohérence ainsi que la technique des votes pondérés, sur laquelle nous nous sommes basés pour développer notre application pour gérer la cohérence des répliques, nous structurons ce document en quatre chapitres:

- Chapitre 1 : Ce chapitre sera consacré aux systèmes distribués à grande échelle.
- Chapitre 2 : Dans ce chapitre nous détaillerons le principe de la réplication et de la cohérence des données dans les SDGE et le principe des votes pondérés.
- Chapitre 3 : Ce chapitre intitulé « Votes pondérés » sera consacré, à la conception de notre application.
- Chapitre 4 : Dans ce chapitre nous détaillerons l'implémentation de notre application et nous discuterons les résultats des expérimentations obtenus.

Chapitre 1

Systèmes distribués à grande échelle

1.1 Introduction

La gestion des données dans un environnement à grande échelle est essentielle pour répondre aux besoins des nouvelles applications. Si la gestion des données dans des systèmes largement distribués a été étudiée ces dernières années, des solutions efficaces et peu coûteuses tardent à émerger en raison de la complexité des problèmes posés par la dynamique environnementale.

Dans ce contexte nous allons présenter quelques notions de base sur le système distribué, ainsi que celles des SDGE (intérêts, inconvénients et caractéristiques).

1.2 Système distribué

Un système distribué est défini comme étant un ensemble des ressources physiques et logiques géographiquement dispersées et reliées par un réseau de communication dans le but de réaliser une tâche commune. Cet ensemble donne aux utilisateurs une vue unique des données du point de vue logique. [10]

D'autre part un système distribué est un groupe d'ordinateurs travaillant ensemble pour apparaître comme un seul ordinateur à l'utilisateur final. [13]

1.3 Définition d'un système distribué à grande échelle

En différencié un SDGE d'un système distribué ordinaire par sa dimension.

La dimension d'un SDGE est mesurée en fonction de plusieurs facteurs : taille du code, quantité de données stockées, accédées et manipulées, connexions et interdépendance entre les composants

du système, hétérogénéité et/ou interopérabilité du ou des réseaux utilisés, intégrité des données, etc.

Pour comprendre ce quoi un SDGE, nous prendrons un exemple typique dans le domaine des banques ou d'assurances où les systèmes d'informations sont distribués et hétérogènes tant sur le plan des architectures matérielles que logicielles (Figure 1). [6]

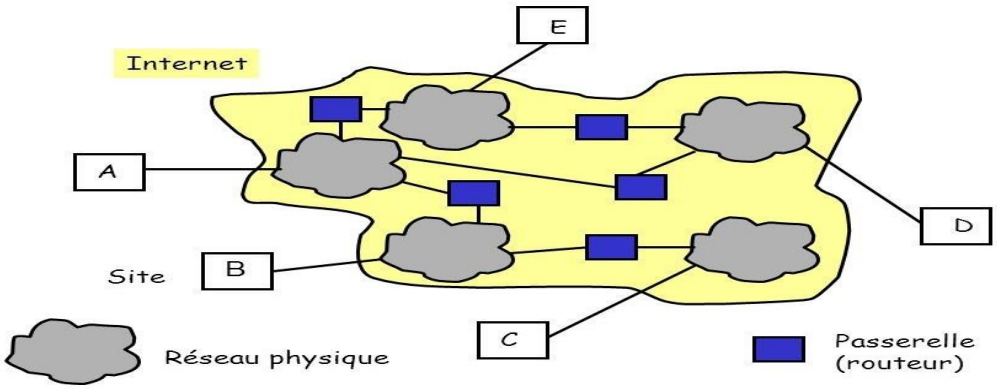


Figure 1 – Système distribué à grande échelle [16]

1.4 Intérêts et inconvénients des systèmes distribués à grande échelle

1.4.1 Intérêts des systèmes distribués à grande échelle

- Mise en commun d'un grand nombre de ressources à faible coûts et avec une puissance globale supérieure à celle des gros calculateurs ;
- Disponibilité et flexibilité c'est-à-dire qu'un composant peut tomber en panne sans bloquer le système et la distribution de la charge ;
- Partage des ressources coûteuses entre plusieurs utilisateurs ;
- Accès aux mêmes ressources depuis tous les endroits du système. [5]

1.4.2 Inconvénients des systèmes distribués à grande échelle

- Les logiciels de gestion de tels systèmes sont difficiles à concevoir ;
- Problèmes inhérents aux communications (Saturation, perte de message, etc.) ;
- Partage et distribution des données qui imposent des mécanismes de sécurité. [5]

1.5 Caractéristiques des systèmes distribués à grande échelle

Un système distribué doit assurer plusieurs propriétés pour être considéré comme performant : la transparence, le passage à l'échelle, la disponibilité et l'autonomie.

- **Transparence**

La transparence permet de cacher aux utilisateurs les détails techniques et organisationnels d'un système distribué ou complexe. L'objectif est de pouvoir faire bénéficier aux applications une multitude de services sans avoir besoin de connaître exactement la localisation ou les détails techniques des ressources qui les fournissent.

Pour un système distribué la transparence a plusieurs niveaux :

– **Accès** : cacher l'organisation logique des ressources et les moyens d'accès à une ressource ;

– **Migration** : une ressource peut changer d'emplacement sans que cela ne soit aperçu ;

– **Réplication** : les ressources sont dupliquées mais les utilisateurs n'ont aucune connaissance de cela ;

– **Panne** : si un nœud est en panne, l'utilisateur ne doit pas s'en rendre compte et encore moins de sa reprise après panne ;

– **Concurrence** : rendre invisible le fait qu'une ressource peut être partagée ou sollicitée simultanément par plusieurs utilisateurs.

- **Passage à l'échelle**

Le concept de passage à l'échelle désigne la capacité d'un système à continuer à délivrer avec un temps de réponse constant un service même si le nombre de clients ou de données augmente de manière importante. Le passage à l'échelle peut être mesuré avec au moins trois dimensions :

– Le nombre d'utilisateurs et/ou de processus (passage à l'échelle en taille) ;

– La distance maximale physique qui sépare les nœuds ou ressources du système (passage à l'échelle géographique) ;

– Le nombre de domaines administratifs (passage à l'échelle administrative).

- **Disponibilité**

– Un système est dit disponible s'il est en mesure de délivrer correctement le (les services) de manière conforme à sa spécification ;

– Pour rendre un système disponible, il faut donc le rendre capable de faire face à tout obstacle qui peut compromettre son bon fonctionnement ;

– Parmi les causes qui peuvent rendre un système indisponible nous pouvons citer :

  - Pannes qui sont des conditions ou évènements accidentels empêchant le système, ou un de ses composants, de fonctionner de manière conforme à sa spécification ;
  - Les surcharges qui sont des sollicitations excessives d'une ressource du système entraînant sa congestion et la dégradation des performances du système ;
  - Les attaques de sécurité qui sont des tentatives délibérées pour perturber le fonctionnement du système, engendrant des pertes de données et de cohérences ou l'arrêt du système.

Pour faire face aux pannes, deux solutions sont généralement utilisées :

– La première consiste à détecter la panne et à la résoudre, et ce dans un délai très court. La détection des pannes nécessite des mécanismes de surveillance qui s'appuient en général sur des timeouts ou des envois de messages périodiques entre ressources surveillées et ressources surveillantes ;

– La deuxième solution consiste à masquer les pannes en utilisant la réplication. Ainsi, quand une ressource est en panne, le traitement qu'elle effectuait est déplacé sur une autre ressource disponible. La réplication peut être aussi utilisée pour faire face à la seconde cause d'indisponibilité d'un système (surcharge du système). Pour réduire la surcharge d'une ressource, les tâches sont traitées parallèlement sur plusieurs répliques

ou sur les différentes répliques disponibles à tour de rôle. Une autre technique qui permet de réduire la surcharge d'une ressource consiste à distribuer les services (ou les données) sur plusieurs sites et donc de les solliciter de manière parallèle.

- **Autonomie**

Un système ou un composant est dit autonome si son fonctionnement ou son intégration dans un système existant ne nécessite aucune modification des composants du système hôte. L'autonomie des composants d'un système favorise l'adaptabilité, l'extensibilité et la réutilisation des ressources de ce système. Par exemple, une ressource autonome peut être remplacée avec une autre ressource plus riche en termes de fonctionnalités, ce qui étend les services du système. [6]

1.6 Conclusion

Dans ce chapitre nous nous sommes intéressés aux SDGE. Ils offrent de hautes performances de calcul mais à coût élevé. D'autre part, les SDGE offrent des ressources aux utilisateurs connectés aux systèmes (internet, grille, etc.) pour la résolution de grands projets scientifiques.

Le SDGE exploite les puissances de calcul de toutes ses ressources et les combine en synergie, créant en quelque sorte un supercalculateur virtuel extrêmement puissant.

Dans le prochain chapitre nous présenterons la réplication et la cohérence des données dans les systèmes distribués.

Chapitre 2

Réplication et cohérence des données

2.1 Introduction

Les chercheurs ont toujours essayé de fournir plus de performances pour leurs systèmes afin de réduire les coûts de connexion et d'augmenter la puissance de calcul, ils ont donc déjà étudié le partage de données dans les réseaux à grande échelle ces dernières années et ont découvert que la réplication des données nécessite de conserver son contenu cohérent. Par conséquent, un traitement supplémentaire est nécessaire lors des mises à jour, c'est pourquoi chaque modification doit être publiée sur toutes les répliques des mêmes données pour assurer la cohérence, ce qui aura pour effet de générer des coûts de traitement supplémentaires. Dans ce contexte, il faut mettre en place le modèle de gestion de la cohérence bien adapté à la stratégie de réplication, qui vise à présenter différents compromis entre les garanties en termes de cohérence et d'efficacité de mise en œuvre.

Dans ce chapitre nous allons présenter la réplication, la cohérence des données et le vote pondéré dans les SDGE.

2.2 Réplication

La réplication des données est un outil très efficace utilisé dans les systèmes distribués, pour donner plus de disponibilité aux données et en conséquence plus de performance d'accès aux données. La réplication consiste à créer des copies d'une donnée et les stocker dans des endroits différents dans le réseau. Les principaux objectifs de la réplication sont l'amélioration des performances des requêtes sur les données locales et surtout l'augmentation de la disponibilité des données. Les lectures sont exécutées sur le nœud disposant de la copie la plus proche du client, ce qui diminue le nombre de transfert de données. Quand à la disponibilité des données, la réplication

permet de ne plus dépendre d'un nœud central unique, et utiliser une copie sur un nœud lorsqu'elle est indisponible sur un autre. [7]

2.2.1 Protocoles de Réplication

Dans les systèmes distribués à grande échelle, les deux protocoles de réplication de références sont les réplications active, passive.

2.2.1.1 Réplication Passive

Dans ce protocole, une seule copie, appelée copie primaire, reçoit la requête d'un client et l'exécute. Les autres copies, nommées copies secondaires ou copies de sauvegardes, ne sont là que pour prendre le relais en cas de défaillance de copie primaire. Afin d'assurer la cohérence des copies secondaires, la copie primaire leur diffuse régulièrement son nouvel état (un point de reprise). Cette méthode assure que toutes les copies ont la même valeur même si les traitements sont non déterministes car il y a diffusion de l'état de la copie primaire vers les copies secondaires et non-exécution des requêtes (Figure 2). [7]

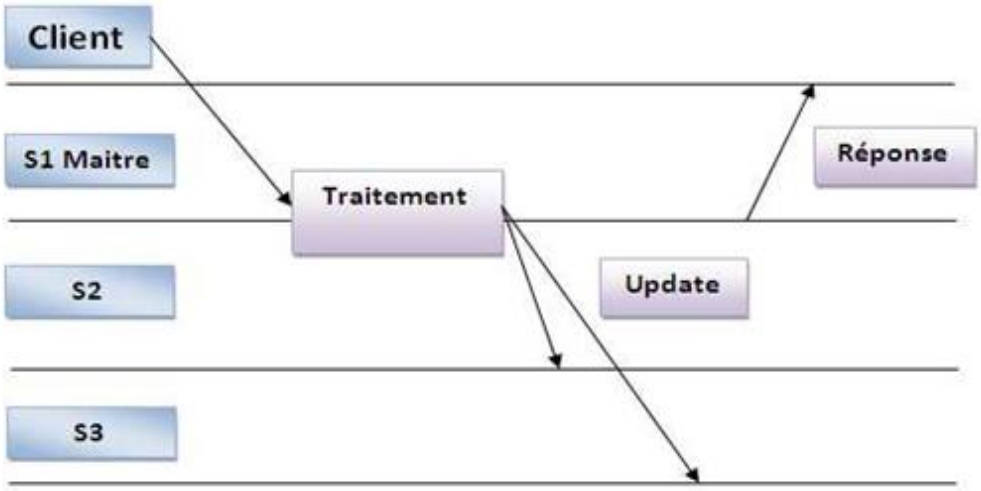


Figure 2 – Protocole de réplication passive [8]

2.2.1.2 Réplication Active

Dans ce protocole, toutes les copies jouent le même rôle. Elles reçoivent toutes la même séquence totalement ordonnée de requêtes de la part des clients, les exécutent de manière déterministe et renvoient la même séquence totalement ordonnée des réponses (Figure 3). [7]

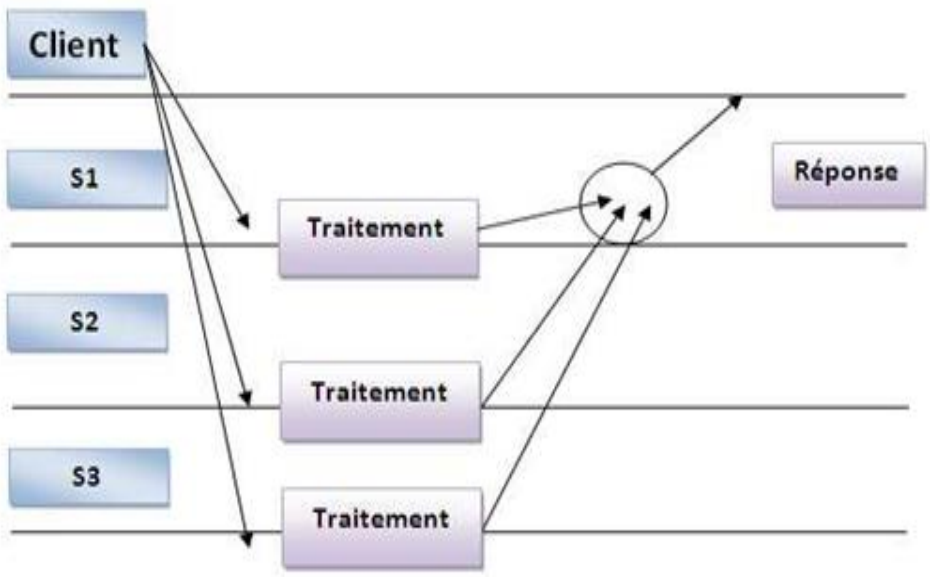


Figure 3 – Protocole de réplication active [8]

Ces deux protocoles de réplication sont des protocoles de base où deux autres protocoles en dérivent : protocoles de réplication Semi-active et semi-passive.

2.3 Cohérence

Le fait de répliquer les données dans des emplacements physiques différents engendre un problème de cohérence auquel doit faire face le système. Lorsqu'une donnée d'une des répliques est modifiée, les autres répliques deviennent obsolètes tant que la mise à jour ne leur est pas parvenue. Par conséquent, il est important de mettre en place une stratégie de réplication qui permet de gérer en toute sécurité des opérations d'écriture/lecture de type CRUD (Create, Read, Update, Delete).

En effet, la stratégie de réplication affecte le comportement des systèmes distribués en ce qui concerne les propriétés de cohérence, disponibilité et tolérance au partitionnement. Nous savions

d'après le théorème du CAP (Consistency, Availability, Partition Tolerance) qu'un système distribué ne peut respecter les trois propriétés à la fois. Ce théorème énonce que tout système distribué peut répondre à une seule contrainte parmi la cohérence et la disponibilité. Par conséquent, les hypothèses traditionnelles, telles que la réplication complète ou le support des transactions, doivent être assouplies. Les solutions existantes diffèrent quant au degré de cohérence des données qu'elles fournissent, un compromis reste à déterminer entre la latence, la cohérence et la disponibilité des données.

Le problème de la cohérence des données se pose quand une mise à jour pour une des répliques se produit. Par conséquent, la cohérence des données est principalement influencée par la stratégie de réplication de données adoptée par le système. [11]

2.3.1 Modèles de cohérence

Il existe de nombreux modèles de cohérence, tous n'offrent pas les mêmes performances et n'imposent pas les mêmes contraintes aux programmeurs d'applications.

Nous avons distingué deux grands modèles : la cohérence forte et la cohérence relâchée (ou faible).

2.3.1.1 Modèle de cohérence forte

Les modèles de cohérence forte n'impliquent pas l'utilisation d'opérations spécifiques. Les applications accèdent aux données et le système gère les transferts et copies de manière complètement transparente. Les modèles de cohérence fortes se distinguent les uns des autres par les garanties offertes en termes de cohérence de données.

Il y a plusieurs modèles de cohérence forte par exemple le modèle strict.

- **Cohérence stricte (atomic consistency)**

  La cohérence stricte correspond à l'intuition naturelle de la notion de cohérence. Dans ce modèle, toute lecture retournera la dernière valeur qui a été écrite. C'est le modèle de cohérence qui est généralement mis en œuvre au niveau des unités de gestion mémoire dans les systèmes monoprocesseurs.

2.3.1.2 Modèle de cohérence faible

Les modèles de cohérence faible (relâchée) imposent aux programmeurs l'utilisation de primitives supplémentaires, généralement des primitives de synchronisation. Les informations fournies par ces primitives permettent aux protocoles de cohérence implémentant ces modèles d'être plus efficaces en diminuant le nombre de messages sur le réseau. Aussi, ces modèles sont souvent utilisés par les applications de calcul scientifique à haute performance.

Il y a plusieurs modèles de cohérence relâchée par exemple le modèle faible.

- **Cohérence faible (weak consistency)**

  Ce modèle propose deux types d'accès : les accès ordinaires (lectures/écritures) et les accès de synchronisation (accès aux objets de synchronisation). Les écritures ne sont pas propagées lors des accès ordinaires. En revanche, lors des accès de synchronisation, toutes les informations sont mises à jour. Le modèle garantit que l'ensemble de la mémoire est cohérent à chaque point de synchronisation, c'est-à dire :

  1) Que toutes les modifications locales sont propagées aux autres processus ;
  2) Que toutes les modifications des autres processus sont visibles localement.

2.3.2 Approches de gestion de la cohérence

Deux familles d'approches sont utilisées pour maintenir la cohérence de répliques : les approches pessimistes et les approches optimistes.

2.3.2.1 Approche pessimiste

L'approche qui utilise des protocoles pessimistes interdit tout accès à une réplique à moins qu'elle soit à jour, ce qui donne l'illusion aux utilisateurs qu'ils disposent d'une seule copie consistante. L'avantage principal de cette approche est que toutes les répliques convergent en même temps, ce qui permet de garantir une cohérence forte, évitant ainsi tout problème de divergence. Ce type d'approche est bien adapté pour des systèmes de petite et moyenne échelle. Par contre, elle sera très difficile à mettre en œuvre pour des systèmes à grande échelle.

Plusieurs protocoles pessimistes ont été définis dans la littérature, parmi lesquels nous pouvons citer les protocoles ROWA, ROWAA et Quorum.

ROWA (Read One Write All) : consiste à écrire dans toutes les copies et lire une seule copie. Ce protocole garantit que toutes les copies sont identiques à tout moment, mais l'inconvénient de ce protocole est que tout le système va être bloqué si un nœud tombe en panne jusqu'à ce que la panne soit réparée.

ROWAA (Read Once Write All Available) : Pour remédier au problème précédent, le protocole ROWAA propage les mises à jour uniquement sur les copies qui ne sont pas en panne. Quand une copie reprend sa disponibilité, elle doit d'abord se synchroniser pour effectuer les modifications manquantes. Toutes les copies disponibles sont mises à jour de manière synchrone et les copies indisponibles sont mises à jour de manière asynchrone.

QUORUM : C'est un protocole de cohérence pessimiste, dans lequel les mises à jour se font de manière synchrone sur un sous-ensemble de copies. Ce sous-ensemble forme ce que l'on appelle un quorum en écriture. Pour les lectures, elles se font sur un autre sous-ensemble, appelé quorum de lecture. Une requête d'écriture n'est validée que lorsque au moins *((N/2) +1)* copies sont mises à jour. Lors d'une requête de lecture, il est alors nécessaire de consulter au moins *(N/2)* copies, où '*N'*est le nombre de répliques dans le système. [8]

Les quorums consistent sous ensemble appeler quorums à consensus :

Quorums à consensus :

Dans une réplication de données basée sur le quorum, il est nécessaire qu'une exécution d'une opération (c'est-à-dire une propagation d'une opération de mise à jour ou d'une opération de lecture) soit validée si et seulement si un nombre suffisamment grand de serveurs reconnaît la fin réussie de cette opération. Notons :

N : un nombre de serveurs stockant des copies de données (répliques) ;

R : un entier appelé quorum de lecture, ce qui signifie qu'au moins '*R*' copies ont été lues avec succès ;

W : un entier appelé quorum d'écriture, ce qui signifie que les propagations vers au moins '*W'* serveurs se sont terminées avec succès.

Les relations suivantes sont valables entre *N, R* et *W* :

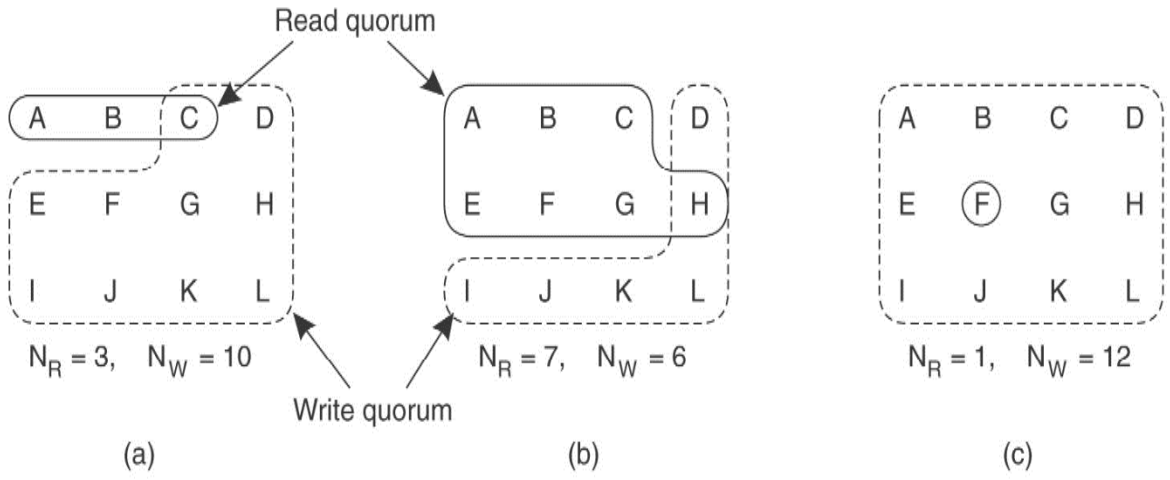
C1 : $W > N / 2$,

C2 : $R + W > N$.

Pour valider une opération de lecture, un serveur doit collecter le quorum de lecture et pour valider une opération d'écriture, il doit collecter le quorum d'écriture. La condition *C1* garantit que la majorité des copies est mise à jour, et *C2* que parmi les copies lues, au moins une est à jour.

L'objectif des algorithmes de consensus est de permettre à un ensemble de serveurs de traiter les commandes des utilisateurs (mises à jour et lectures) tant que le nombre de serveurs actifs n'est pas inférieur à max {*W, R*}. Cela signifie que le système est capable de survivre aux pannes de certains de ses serveurs. [14]

Par exemple dans la figure 4, le nombre de quorum *n=12*. Dans figure 4(a), où le nombre de quorum d'écriture *w=10*, il est impossible d'effectuer deux opérations d'écritures simultanément. Par contre dans figure 4(b), il peut y avoir un conflit entre deux écritures, puisque chaque écriture peut acquérir les 6 permissions nécessaires *(6+6 =12)*. Dans la figure 4(c), on peut lire d'une seule copie mais il faut avoir toutes les permissions pour écrire (ROWA : Read One Write All).



Choix correct | Mauvais choix | Choix correct

Figure 4 – Exemple illustratif pour les quorums [2]

2.3.2.2 Approche optimiste

A la différence de l'approche pessimiste, les protocoles de l'approche optimiste autorisent l'accès à n'importe quelle réplique et ce à tout instant. De cette manière, les copies peuvent diverger et un utilisateur peut donc observer des copies d'un même objet avec des valeurs différentes.

De cette manière, il est alors possible d'accéder à une réplique qui n'est pas nécessairement cohérente, ce qui fait que cette approche tolère une certaine divergence entre répliques. En présence de divergence, ce type d'approche nécessite une phase de détection de divergence entre répliques puis une phase de correction de cette divergence, pour ramener les répliques vers un état cohérent. Bien qu'elle ne garantisse pas une cohérence forte comme dans le cas pessimiste, l'approche optimiste possède néanmoins un certain nombre d'avantages que nous pouvons résumer comme suit :

- Elle améliore la disponibilité car l'accès aux données n'est jamais bloqué ;
- Elle permet aux sites et aux utilisateurs de rester autonomes, etc. [8]

2.4 Votes pondérés

C'est un vote pondéré mais sans critère spécifique. Il est souvent suffisant si le problème étudié n'est pas complexe.

Chaque membre du groupe choisit les causes ou les solutions les plus importantes à ses yeux et les classent par ordre d'importance (en attribuant par exemple le poids 3 pour celle qui lui paraît la plus importante, le poids 2 pour la suivante, etc.).

Nous avons additionné les points de tous les membres, et nous avons retient l'option qui présente le total le plus haut ou bas selon le choix. [4]

Dans un nouvel algorithme de gestion des données répliquées, chaque copie d'un répliqué se voit attribuer un certain nombre de votes. Chaque transaction recueille un quorum de lecture de *'r'* votes pour lire un fichier et un quorum d'écriture de *'w'* votes pour écrire un fichier, de sorte que $r+w$ soit supérieur au nombre total de votes attribués au fichier. Cela garantit qu'il existe une intersection non nulle entre chaque quorum de lecture et chaque quorum d'écriture. Les numéros de version permettent de déterminer les copies actuelles.

Les «*votes*» attribués à chaque copie du dossier sont son poids. Ce modèle offre un bon moyen de généraliser les données répliquées. Il pourrait décrire un modèle primaire / secondaire ou mettre l'accent sur la garantie que les systèmes critiques disposent de copies des données. Le modèle permet même de mettre en cache des nœuds sans poids.

La clé de cette approche est que le quorum de lecture est configuré de sorte qu'il soit suffisamment grand pour qu'au moins une copie de l'ensemble de lecture contienne les données actuelles. Ceci est accompli en s'assurant que la combinaison du quorum de lecture et du quorum d'écriture représente un nombre (poids) qui est plus grand que la somme totale de tous les poids dans le système. Le défi dans un système comme celui-ci est que le choix de ces valeurs déterminera la fiabilité du système face à une panne. L'auteur n'entre pas dans les détails sur les types de pannes qui peuvent survenir, mais il est évident que l'une des copies répliquées devient indisponible. Un autre exemple est celle des partitions réseau où un groupe de répliques existe d'un côté de la partition et un groupe différent existe dans une partition distincte. [12]

Exemple : Nous avons pris 5 banques comme la figure ci dessous.

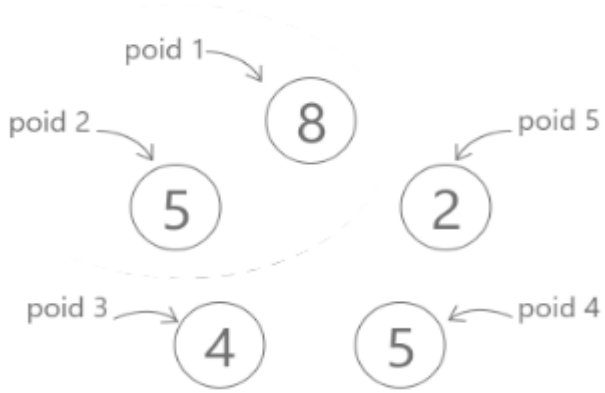


Figure 5 – Exemple des données répliquées dans 5 banques avec leur poids

Dans la figure au-dessus schématise 5 copies d'une même donnée distribué sur 5 banques. Pour modifier une copie nous utilisons les contraintes C1 et C2.

Pour indiquer le quorum, les deux contraintes suivantes doit être vérifier :

La contrainte 1 : somme de quorum de lecture *'r'* avec le quorum d'écriture *'w'* soit supérieur à la somme des poids *'totalweight'*.

La contrainte 2 : le quorum d'écriture *'w'* soit supérieur ou égale à le poids de mise à jour '*updateweight'*.

$$totalweight = \sum pi \qquad \forall\ 1<=i<=n$$

$$updateweight = roundsup\ ((totalweight\ +1)/2)$$

C1': $r + w > totalweight$

C2': $w \geq updateweight$

roundsup(x) arrondit un nombre en entier supérieur.

Soit :

pi : poids de nœud *i*.

r : quorum de lecture.

w : quorum d'écriture.

Dans cet exemple désigné : *totalweight = 24*, *updateweight = 13*, nous prenons au moins

Poids = 13 pour le rédacteur et *Poids = 12* pour le lecteur les schémas suivants illustrent ceci: Figure 6.

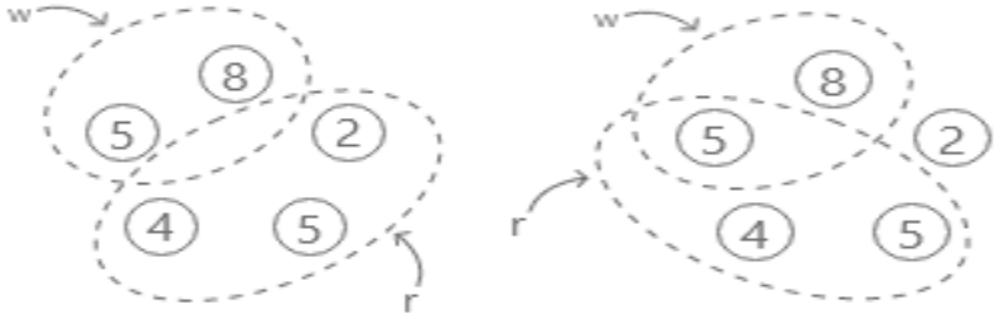


Figure 6 – Exemple de la formation des quorums de lecture et d'écriture

2.5 Conclusion

Dans ce chapitre, nous avons présenté la réplication, une technique permettant d'améliorer les performances et la tolérance aux pannes dans un système distribué.

Le principal avantage de la réplication est que si les données ne sont plus disponibles, le système peut continuer à fournir ses fonctionnalités avec les données répliquées, ce qui augmente la disponibilité des données et la tolérance aux pannes. En revanche, l'utilisation de cette technique engendrera un surcoût, et la principale difficulté de la réplication est de maintenir la cohérence des différentes copies si une mise à jour a été effectué. Le d'effet majeur de la réplication est de maintenir la cohérence des différentes répliques.

En effet, nous avons parlé dans ce chapitre sur la cohérence des répliques, qui est un problème fondamental lorsque l'accès simultané aux données partagées est autorisé, en risque d'avoir l'incohérence entre les différentes répliques si l'un d'entre eux est modifié.

Dans le chapitre suivant nous allons détailler notre approche que nous allons proposer à l’aide d'utilisation la méthode UML.

Chapitre 3

Votes pondérés

3.1 Introduction

Dans le chapitre précédent, nous avons détaillé quelques principes de la réplication et de la cohérence des données dans les SDGE et le principe des votes pondérés.

Dans ce chapitre nous présenterons la conception de l'approche des votes pondérés, qui nécessite des méthodes permettant de mettre en place ce modèle. Nous avons utilisé le langage de modélisation UML, qui est la méthode la plus utilisée dans la conception orienté objet.

3.2 UML (Unified Modeling Language)

3.2.1 Définition

L'UML est un langage général unifié applicable à différents types de systèmes, des domaines, des méthodes et des processus. Cependant, il peut y avoir des situations dans lesquelles l'utilisateur pourrait vouloir étendre le langage pour une utilisation souple et enrichie tout en ajoutant de nouveaux éléments de construction (stéréotypes, valeurs, contraintes, etc.). [1]

3.2.2 Différentes Type de Diagrammes UML

3.2.2.1 Diagrammes statiques (structurels)

- Diagramme classe ;
- Diagramme d'objet ;
- Diagramme composant ;

- Diagramme déploiement ;
- Diagramme de cas d'utilisation.

3.2.2.2 Diagrammes dynamiques (comportementaux)

- Diagramme d'activité ;
- Diagramme séquence ;
- Diagramme d'état-transition ;
- Diagramme de collaboration. [9]

3.2.3 Définition des diagrammes utilisés

Pour la modélisation des besoins, nous utilisons les diagrammes UML suivants : Diagramme de cas d'utilisation, diagramme de classe, diagramme de séquence.

3.2.3.1 Diagramme de classe (Class Diagram)

Les diagrammes de classes expriment de manière générale la structure statique d'un système, en termes de classes et de relations entre elles. De même qu'une classe décrit un ensemble d'objets, une association décrit un ensemble de liens, les objets sont des instances de classes et les liens sont des instances de relations.

En général un diagramme de classe peut contenir les éléments suivants :

- Les classes ;
- Les associations ;
- Les attributs ;
- Etc. [3]

3.2.3.2 Diagramme de cas d'utilisation (Use Case Diagram)

Un diagramme de cas d'utilisation est un graphe d'acteurs, un ensemble de cas d'utilisation englobés par la limite du système, des associations de communication entre les acteurs et les cas d'utilisation, et des généralisations entre cas d'utilisation.

Il est destiné à représenter les besoins des utilisateurs par rapport au système, identification des acteurs, les acteurs d'un système sont les entités externes à ce système qui interagissent avec lui. [9]

3.2.3.3 Diagramme de séquence (Sequence Diagram)

Le diagramme de séquences représente la succession chronologique des opérations réalisées par des acteurs vis-à-vis du système impliqué dans un cas d'utilisation. Les opérations effectuées sont sous forme de messages qui font passer d'un objet à l'autre. [3]

3.3 Conception

3.3.1 Diagramme de classe

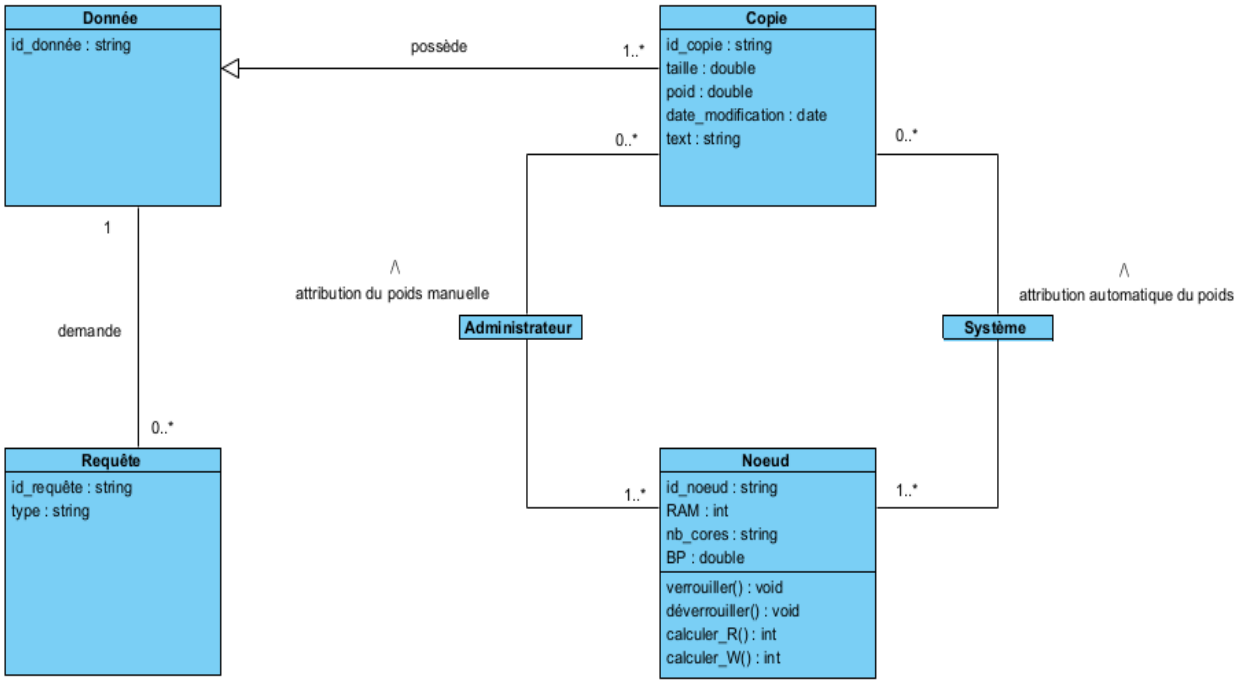


Figure 7 – Diagramme de classe

3.3.2 Diagramme de cas d'utilisation

1. **Système** : pour notre application cette acteur est responsable de la distribution réseau, de la gestion des opérations de lecture et d'écriture et du calcul des quorums *R* et *W*.

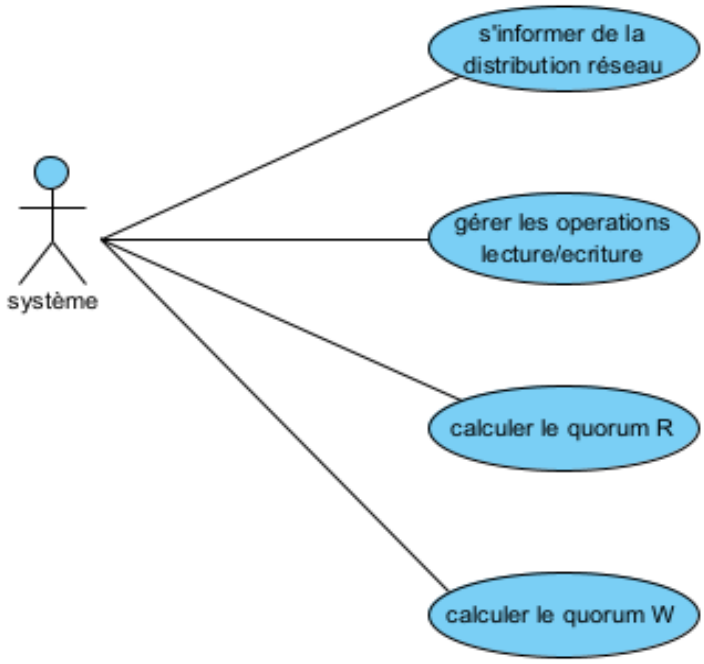


Figure 8 – Diagramme de cas d'utilisation des principales fonctionnalités

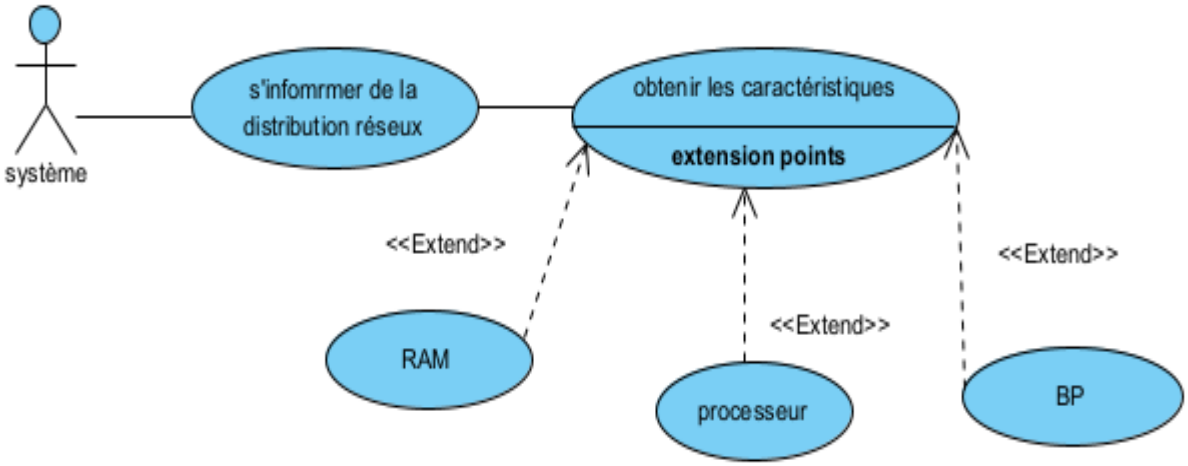


Figure 9 – Diagramme de cas d'utilisation de la distribution réseau

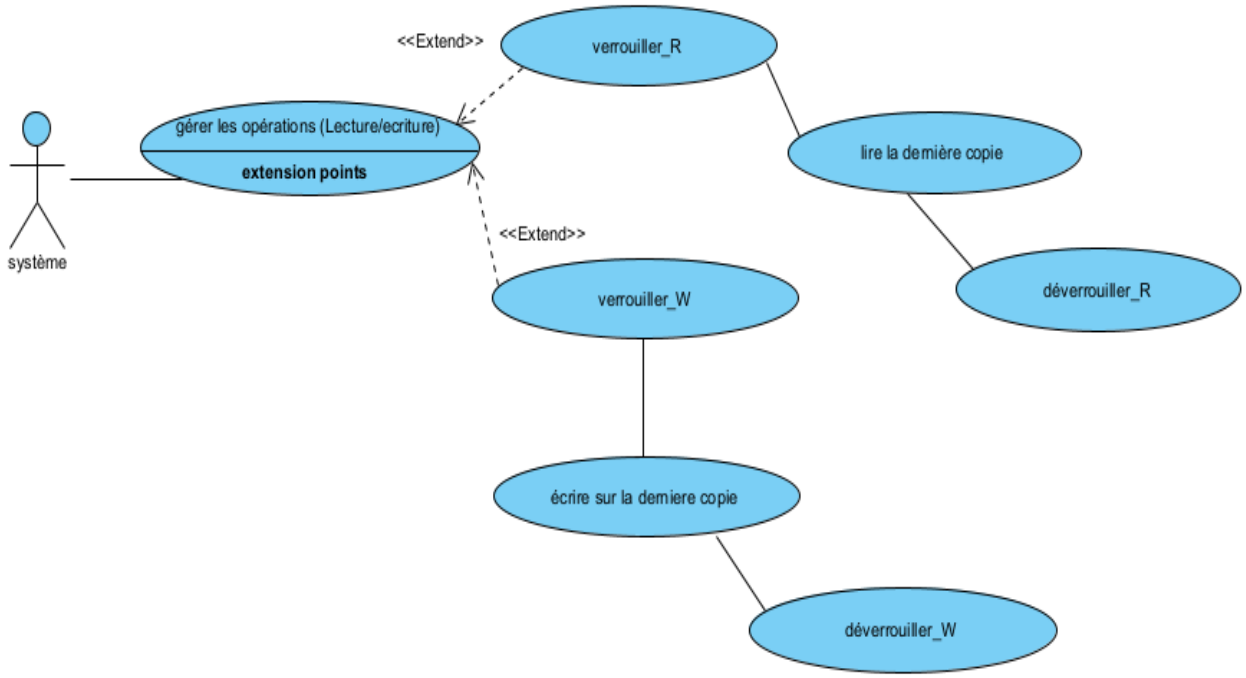


Figure 10 – Diagramme de cas d'utilisation de la gestion des opérations de lectures et d'écriture

- Verrouiller_R : signifie le verrouillage pour lecture de toutes les copies appartenant à l'ensemble du quorum *R* ;
- Déverrouiller_R : signifie la libération de toutes les copies appartenant à l'ensemble du quorum *R* ;
- Verrouiller_W : signifie le verrouillage pour écriture de toutes les copies appartenant à l'ensemble du quorum *W* ;
- Déverrouiller_W : signifie la libération de toutes les copies appartenant à l'ensemble du quorum *W*.

2. **Utilisateur :** cet acteur peut jouer les deux rôles : le rôle du lecteur ou le rôle du rédacteur séparément sachant que notre système doit être l'unique responsable de la gestion de la cohérence des copies.

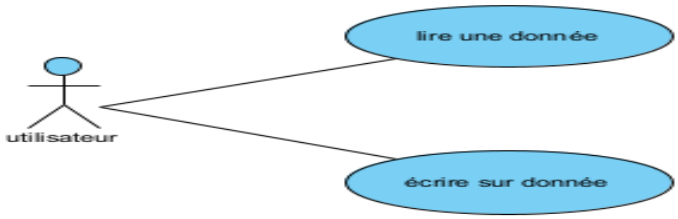


Figure 11 – Diagramme de cas d'utilisation pour l'écriture et la lecture d'une donnée

3.3.3 Diagramme de séquence

3.3.3.1 Diagramme de séquence pour l'administrateur

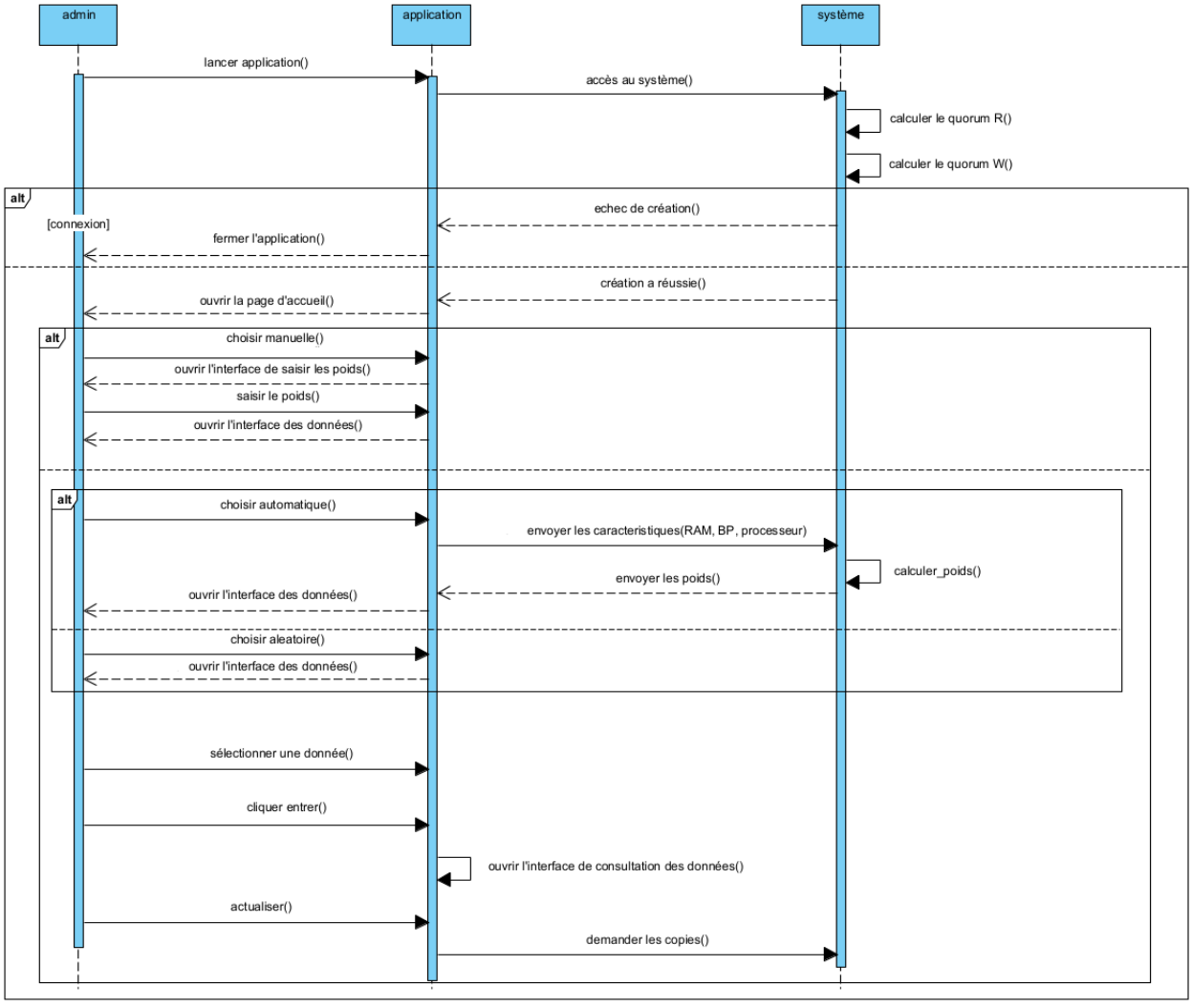


Figure 12 – Diagramme de séquence pour l'administrateur

- Au début l'administrateur lance l'application, une fois que l'application accède au système, ce dernier calcule les quorums W et R ;
- Si l'accès au système échoue, l'application envoie un message à l'administrateur et se termine ;
- Sinon la page d'accueil s'affiche ;
- Si l'utilisateur choisit la méthode manuelle ;
- L'interface de saisie du poids s'affiche et l'utilisateur peut attribuer les poids manuellement. Ensuite l'interface des données s'affiche ;
- Sinon l'administrateur peut choisir entre une attribution automatique ou aléatoire ;
- Le système calcule le poids de chaque copie et envoie le poids à l'application ;
- L'administrateur sélectionne la donnée ;
- L'interface de consultation des données s'affiche ;
- L'application demande les copies au système.

3.3.3.2 Diagramme de séquence pour le lecteur

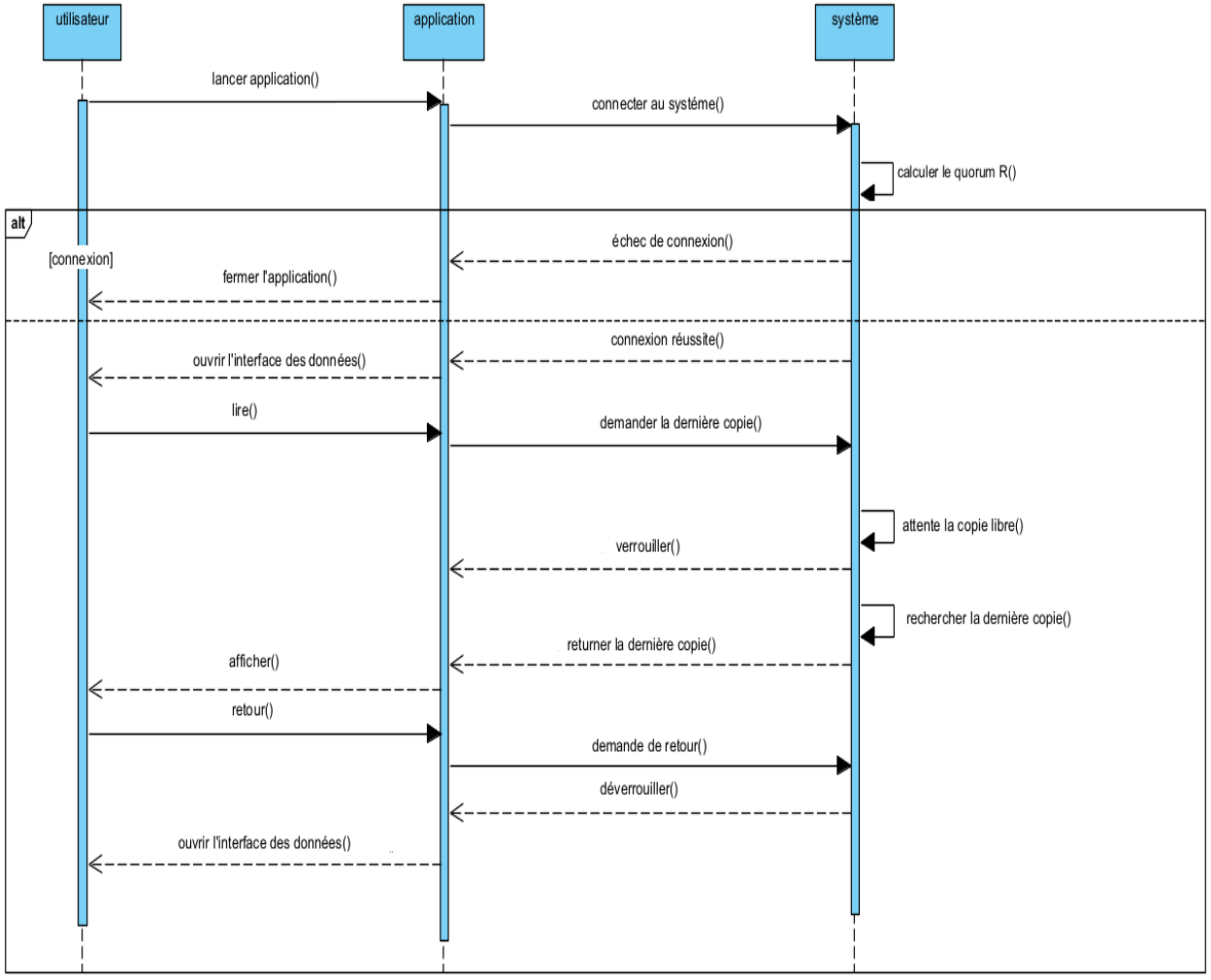


Figure 13 – Diagramme de séquence pour le lecteur

- Au début l'utilisateur lance l'application qui se connecte au système. Ensuite le système calcule le quorum *R* ;
- Si l'application n'est pas connectée au système un message d'erreur apparaît et l'exécution se termine ;
- Sinon l'interface des données s'affiche ;
- L'utilisateur choisis lire et l'application demande au système la dernière copie ;
- Le système attend la réponse des copies libres,

- Une fois le quorum atteint le système verrouille les copies correspondantes ;
- Le système recherche la dernière copie ;
- Le système envoie la dernière copie à l'application ;
- L'identificateur et l'emplacement de la copie sont affichées ;
- Le système déverrouille les copies correspondantes.

3.3.3.3 Diagramme de séquence pour le rédacteur

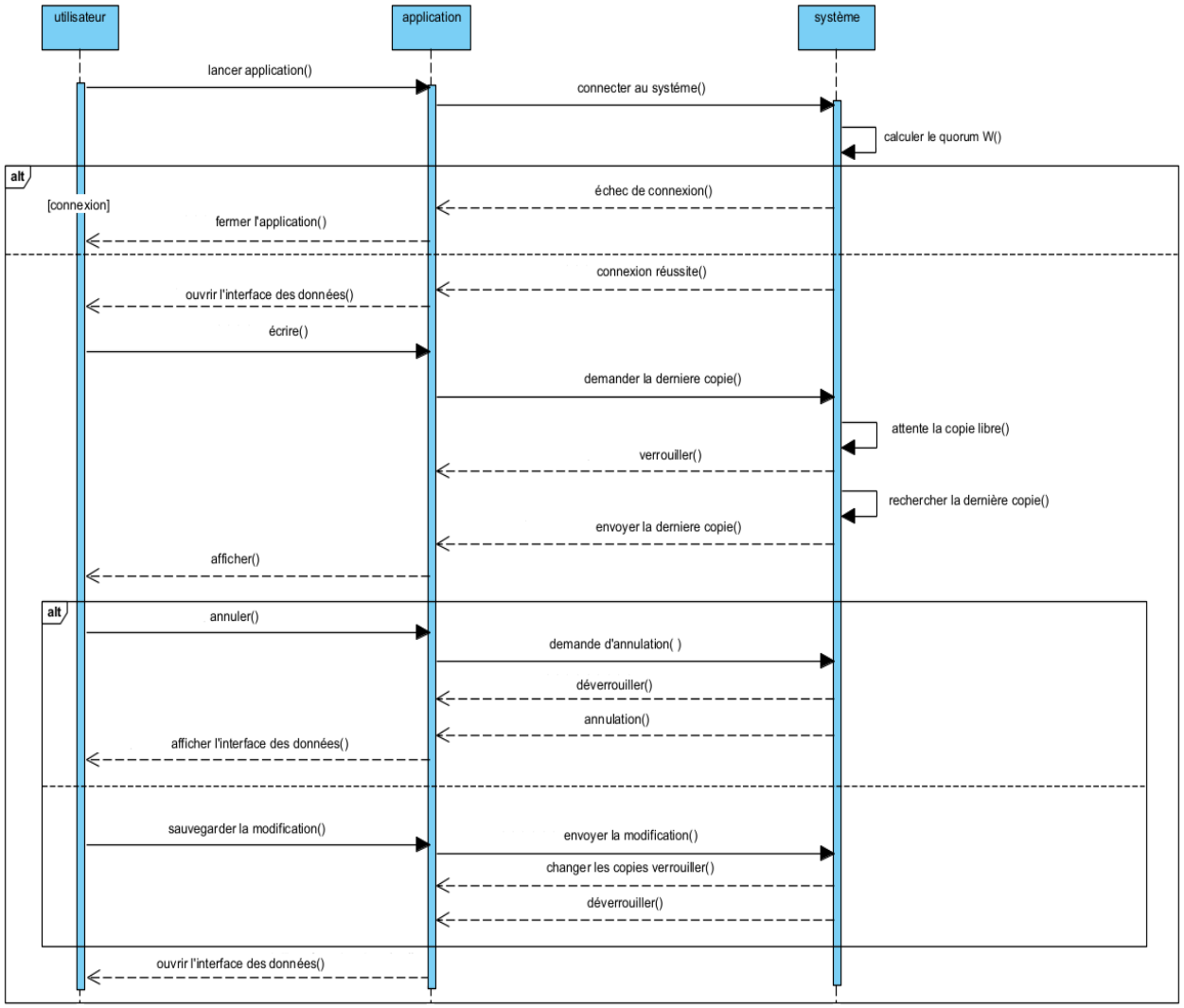


Figure 14 – Diagramme de séquence pour le rédacteur

- Au début l'utilisateur lance l'application qui se connecte au système. Ensuite le système calcule le quorum W ;
- Si l'application n'est pas connectée au système un message d'erreur apparaît et l'exécution se termine ;
- Sinon l'interface des données s'affiche ;
- L'utilisateur choisis écrire et l'application demande au système la dernière copie ;
- Le système attend la réponse des copies libres,
- Une fois le quorum atteint le système verrouille les copies correspondantes ;
- Le système recherche la dernière copie ;
- Le système envoie la dernière copie à l'application ;
- L'identificateur et l'emplacement de la copie sont affichés ;
- Si l'utilisateur choisit d'annuler ;
- Le système déverrouille les copies correspondantes ;
- Sinon l'utilisateur sauvegarde la modification ;
- Le système change les copies correspondantes ;
- Le système déverrouille les copies correspondantes.

3.4 Conclusion

Dans ce chapitre nous avons décrit la conception de l'approche basée sur les votes pondérés, en utilisant les diagrammes de modélisation d'UML.

Dans le chapitre suivant, nous allons implémenter cette approche afin de l'évaluer et d'étudier son comportement.

Chapitre 4

Implémentation

4.1 Introduction

Ce chapitre est une présentation des résultats de la mise en œuvre de notre approche. Nous présenterons des outils de développement de l'approche des votes pondérés.

Dans ce qui suit nous commencerons par la présentation des principales fonctionnalités de nos implémentations et finir par la présentation des résultats obtenus des expérimentations qui nous ont permis de vérifier l'utilité de notre approche dans l'attribution des poids.

4.2 Description de l'environnement de développement

Nous avons développé notre application sur une machine fonctionnant sous le système Windows, avec les caractéristiques suivantes :

- Un processeur Intel ® Celeron® CPU N2840 ;
- D'une fréquence de 2.16 Ghz ;
- Une capacité de mémoire de 4Go.

L'application est développée par le langage de programmation java et pour cela nous avons utilisé l'IDE NetBeans 8.2.

JAVA : est un langage de programmation et une plate-forme informatique. Beaucoup d'applications et de sites Web ne fonctionne pas si Java n'est pas installé et leur nombre ne cesse de croître chaque jour. Java est rapide, sécurisé et fiable. [15]

NetBeans : est un environnement de développement intégré. En plus de Java, NetBeans permet la prise en charge native de divers langages tels le C, le C++, le JavaScript, le XML, le Groovy, le PHP et le HTML, ou d'autres (dont Python et Ruby) par l'ajout de greffons. [17]

4.3 Implémentation

Nous allons présenter dans ce qui suit les différentes fonctionnalités et fenêtres de notre application.

La figure 15 ci-dessous représente la page d'accueil qui est la première interface qui apparaît à l'administrateur, nous remarquons 3 boutons (automatique, manuelle, aléatoire).

Le bouton automatique est lié aux trois choix suivants :

- RAM ;
- Bande passante (BP) ;
- Processeur.

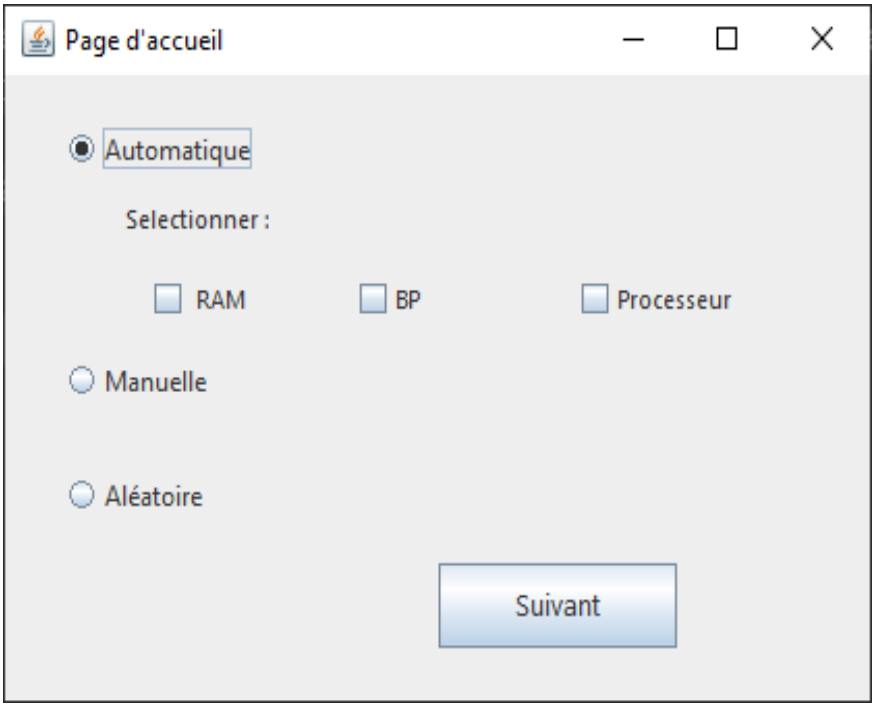


Figure 15 – La page d'accueil de l'application

Si l'administrateur choisis « Automatique » et sélectionne la case RAM ou une autre case ou combine entre toutes les cases au même temps pour consulter les copies, alors la première interface qui apparaît est l'interface des données. Cette dernière affichera les paramètres suivants montrés dans la figure 16.

- Nom fichier ;
- Taille ;
- Chemin.

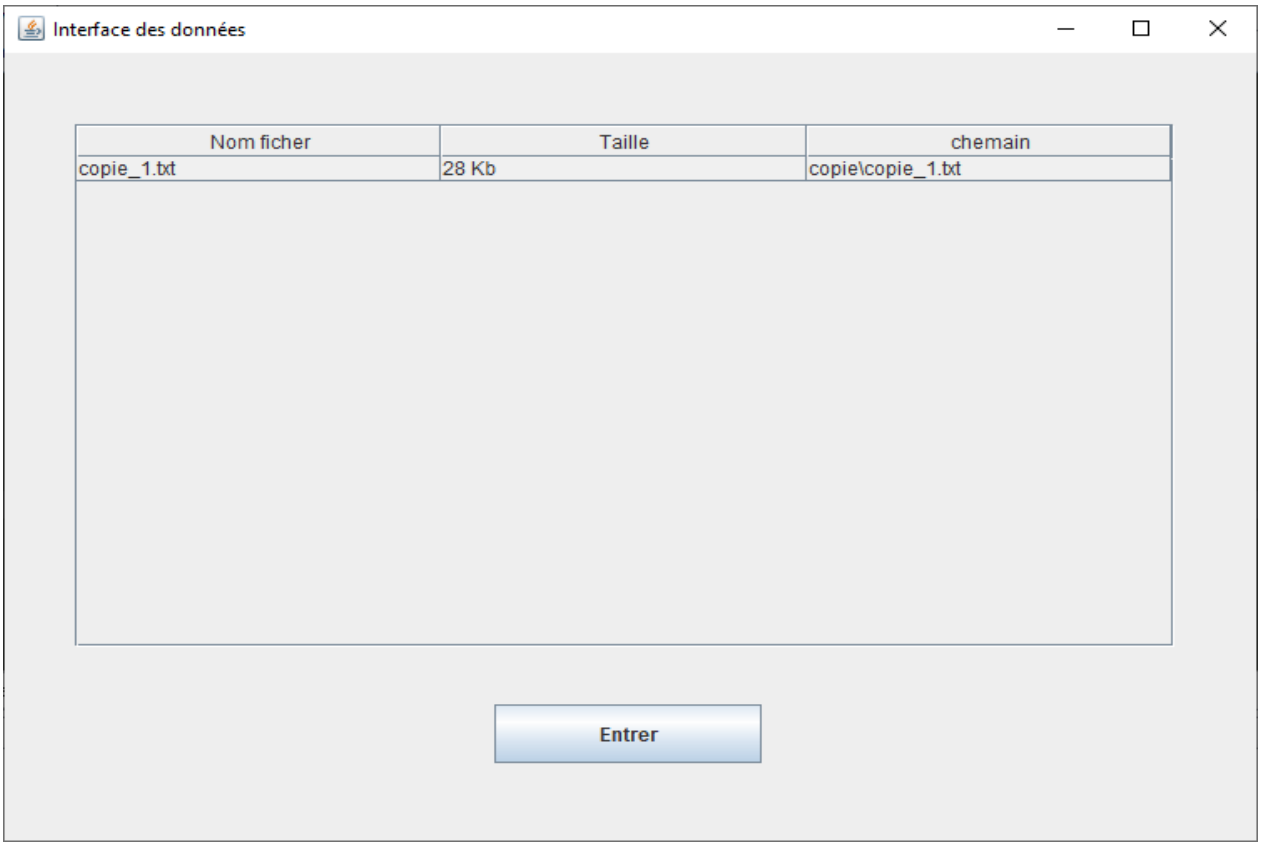


Figure 16 – L'interface des données

Puis il appuie sur le bouton entrer et l'interface de consultation des données s'ouvre affichera les paramètres suivants montrés dans la figure 17.

- Identifiant de la copie ;
- Poids ;

- Text ;
- Date de modification.

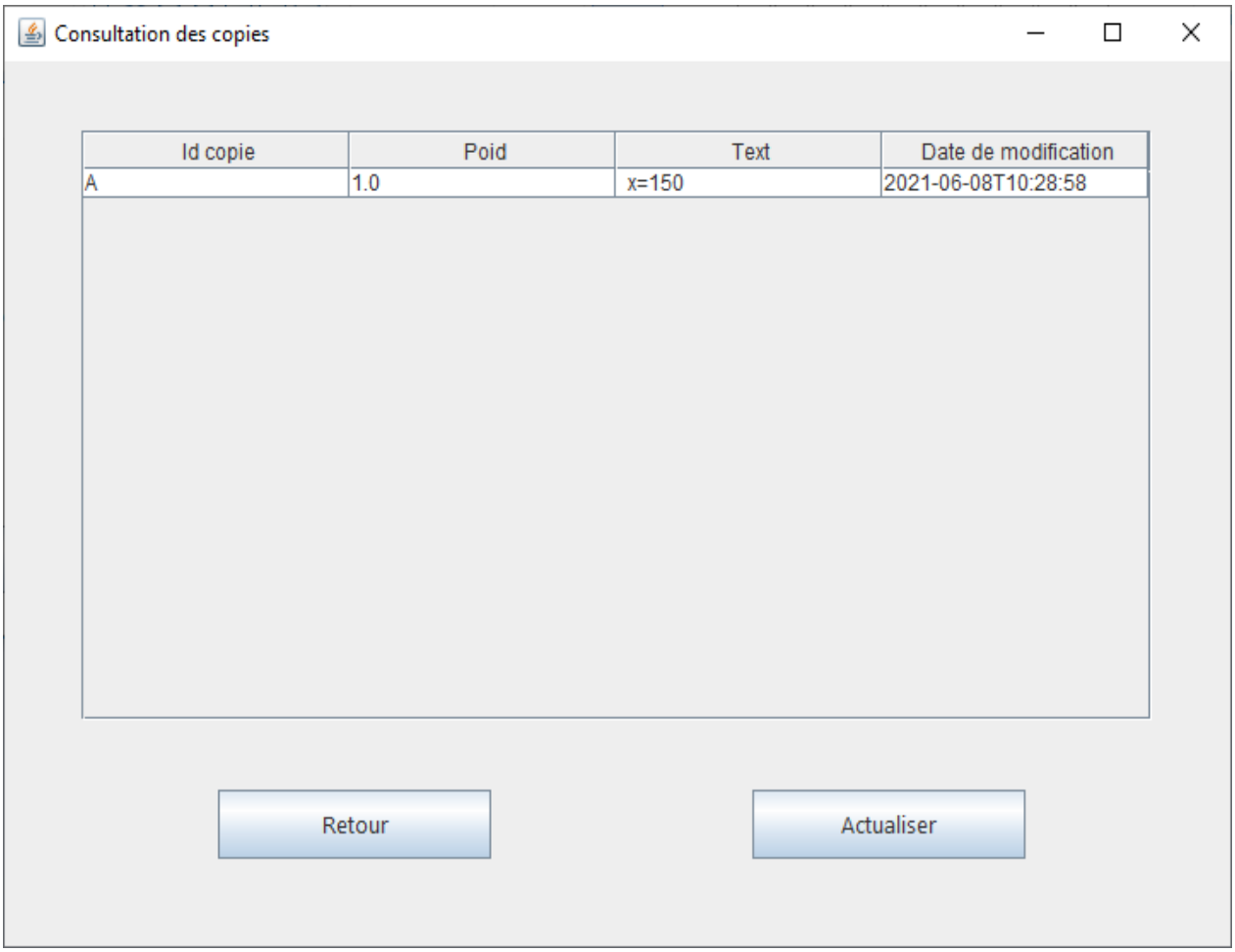


Figure 17 – L'interface de consultation des copies

Sinon si l'administrateur choisis « Manuelle » pour consulter les copies et changer le poids de la copie, alors l'interface qui apparaît affichera les paramètres suivant qui montré dans la figure 18.

- Identifiant de la copie ;
- Poids ;
- Text ;
- Date de modification.

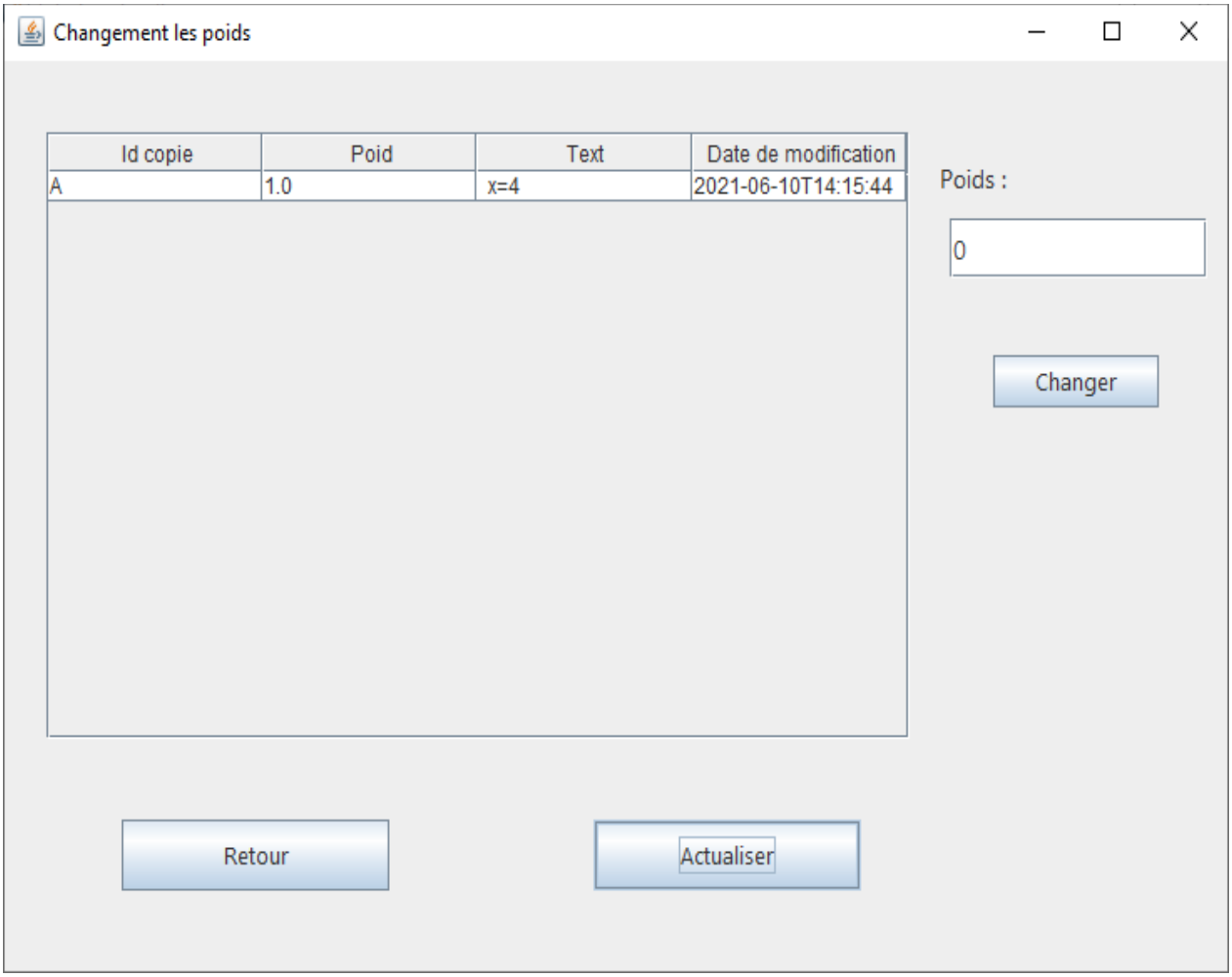


Figure 18 – L'interface de changement des poids

Si l'administrateur choisis « Aléatoire » pour consulter les copies, les mêmes interfaces que sur les figures 16 et 17 s'afficherons.

La figure 19 ci-dessous représente l'interface de lire/écrire qui apparaît à l'utilisateur pour soit lire une donnée ou modifier (écrire) sur une donnée.

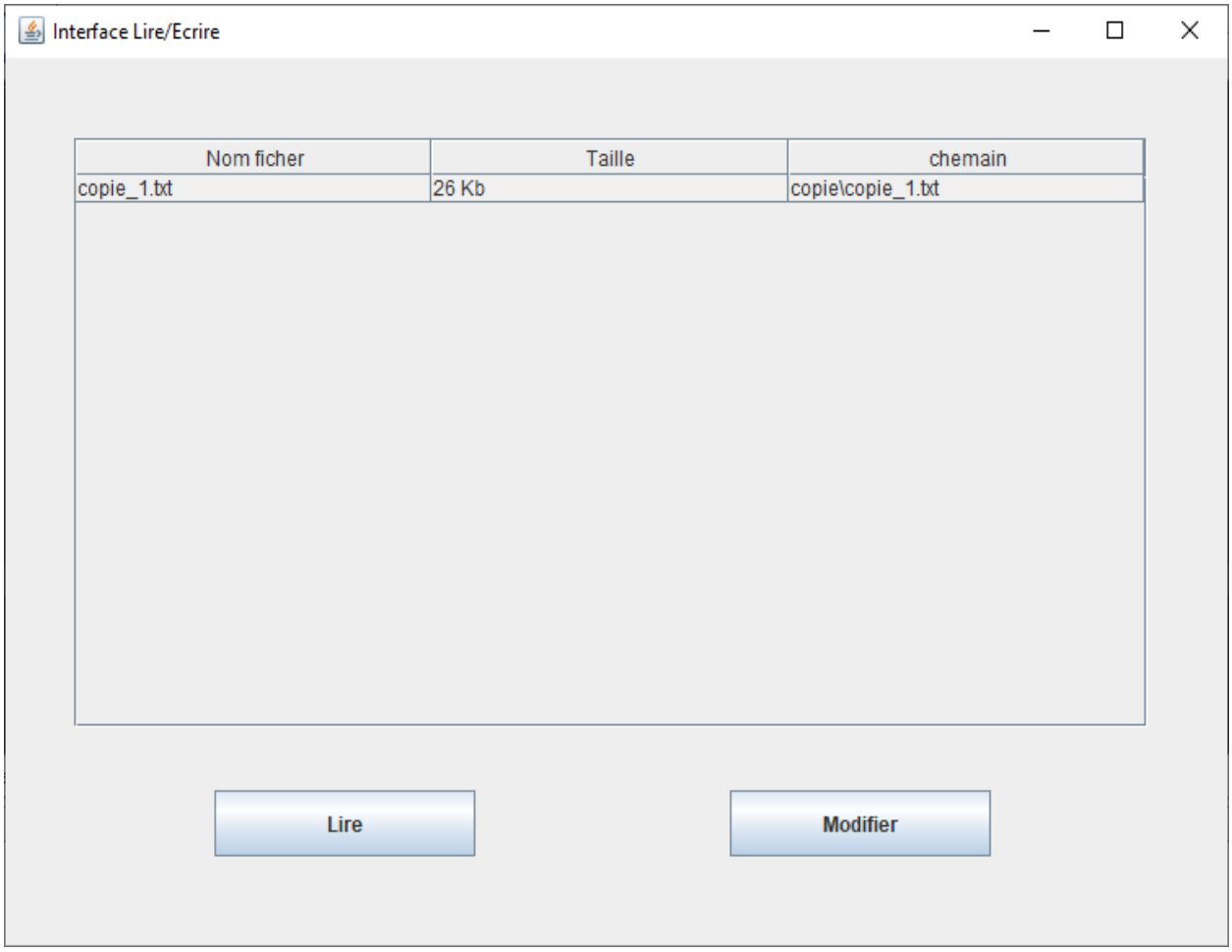


Figure 19 – L'interface de lire/écrire

4.4 Expérimentations et résultats

Nous avons voulu implémenter cette approche pour le but de comment faire distribuer les poids sur les différentes copies pour améliorer les performances les plus gérer. Pour cela nous avons fait les différentes expérimentations.

Les expérimentations se sont effectuées sur 5 nœuds selon les paramètres de simulation suivantes dans le tableau 1 :

Tableau 1 – Paramètres de simulation

	Processeur (GHz)	RAM (GO)	BP (MO/s)
Nœud 1	1.1	4	21
Nœud 2	2.16	4	18
Nœud 3	2.4	8	12
Nœud 4	2.67	6	13
Nœud 5	2.16	8	16

Dans cette section, nous présenterons les résultats obtenus de quelques expérimentations selon différents paramètres d'exécution.

Ces expérimentations ont été effectuées en utilisant approche qui nous a permis d'évaluer nos fonctionnalités proposés (automatique, manuelle et aléatoire).

4.4.1 1ère expérimentation

Dans cette expérimentation nous avons effectué des opérations d'accès aux données (lecture/écritures) en attribuant automatiquement des poids selon les différents paramètres (caractéristiques des nœuds : RAM, processeur, BP).

Après avoir extrait toutes les propriétés, nous allons ranger la RAM et le Processeur avec l'ordre croissant, par contre la bande passante (BP) avec l'ordre décroissant pour déduire des poids logiques.

Tableau 2 – Les poids des nœuds

	Processeur	RAM	BP	Poids
Nœud 1	1.1	4	21	0.25
Nœud 2	2.16	4	18	0.4
Nœud 3	2.4	8	12	0.883333333
Nœud 4	2.67	6	13	0.8
Nœud 5	2.16	8	16	0.666666667
Total	12	10	15	

Pour chaque nœuds i :

$$\sum_{j=1}^{Nombre\ de\ paramètre} Poids_{ij}/Somme_j$$

Ensuite nous avons effectué des opérations d'accès aux données (lecture/écritures) en attribuant automatiquement des poids.

Le tableau 3 présente les temps de réponse moyens par rapport aux nombres de copies pour des opérations de lectures.

Tableau 3 – Temps de réponses moyens par nombre de copies pour lecture

	Nombre de copies	**10**	**15**	**20**	**25**	**Moyenne**
Temps de réponse moyens (ms)	**RAM**	703.6	728.9	778.3	642.8	713.4
	BP	566.45	656.55	715.5	653.15	647.91
	Processeur	581.35	505.35	597.1	585.45	567.31

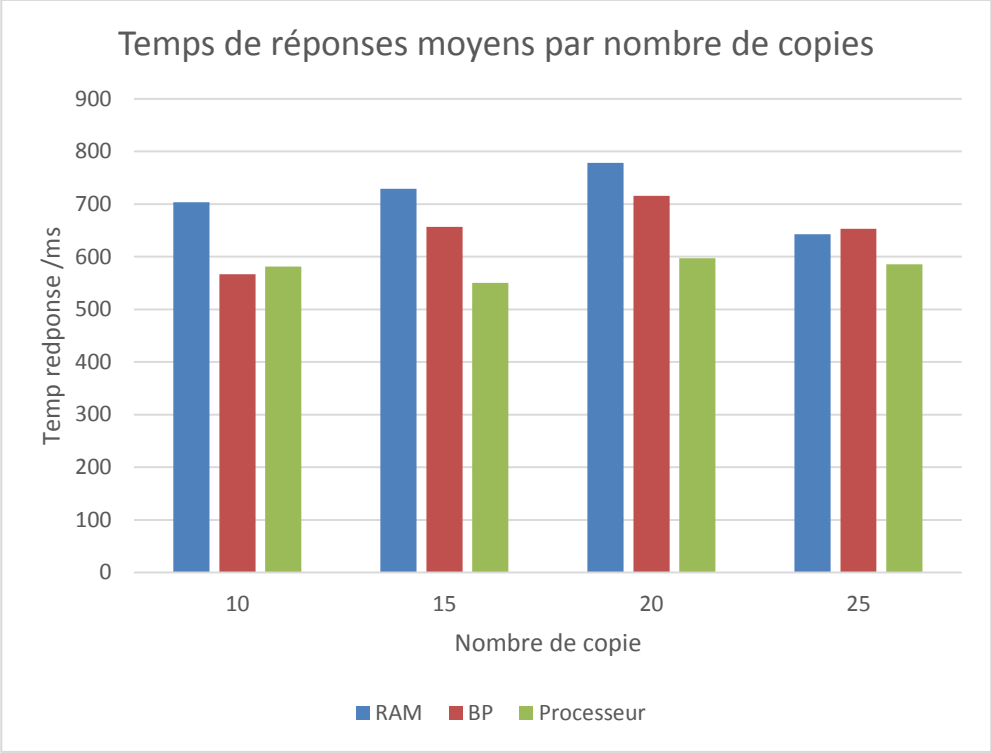


Figure 20 – Temps de réponses moyens par nombre de copies pour lecture

Les résultats obtenus dans le tableau 1 sont modélisés par l'histogramme de la figure 20 ci-dessus, qui représente les temps de réponses moyens par nombre de copies pour lecture.

Nous remarquons que le paramètre processeur est le meilleur parce que les temps de réponses moyens sont les plus faibles.

Le tableau 4 présente les temps de réponse moyens par rapport aux nombres de copies. Pour des opérations d'écritures.

Tableau 4 –Temps de réponses moyens par nombre de copies pour l'écriture

	Nombre de copies	**10**	**15**	**20**	**25**	**Moyenne**
Temps de réponse moyens (ms)	**RAM**	577.55	582.7	575	583.65	579.72
	BP	569.75	576.2	548.9	559.7	563.63
	Processeur	553.45	577.45	554.7	597.25	570.71

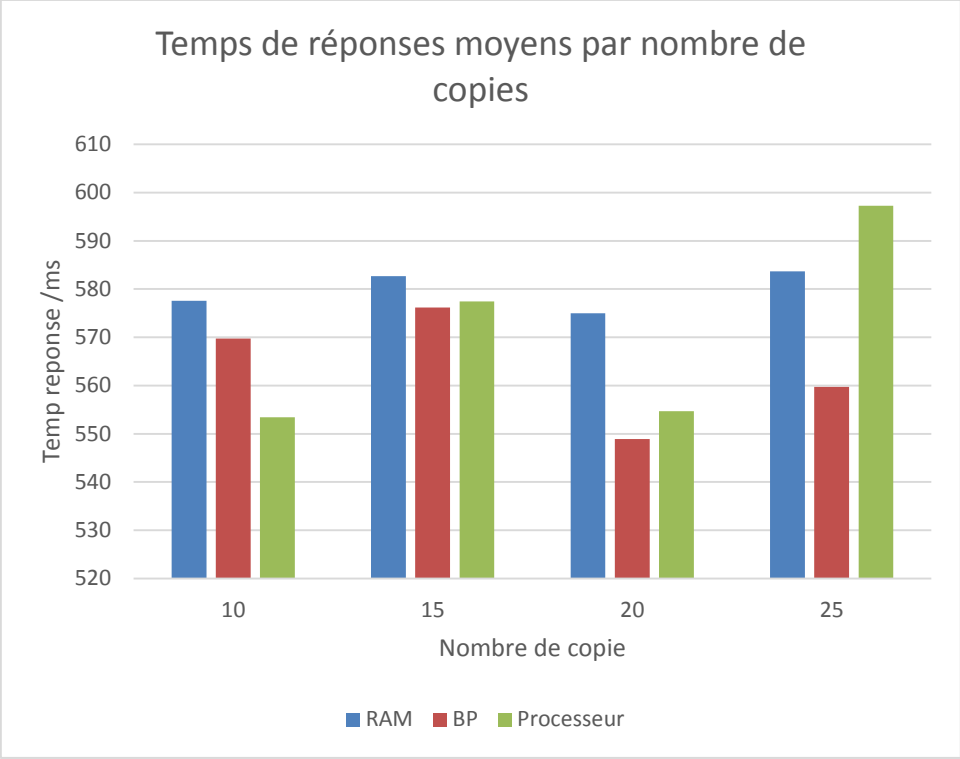


Figure 21 – Temps de réponses moyens par nombre de copies pour l'écriture

Les résultats obtenus dans le tableau 2 sont modélisés par l'histogramme de la figure 21 ci-dessus, qui représente les temps de réponses moyens par nombre de copies pour l'écriture.

Nous remarquons que le paramètre bande passante est le meilleur parce que les temps de réponses moyens sont les plus faibles.

4.4.2 2ème expérimentation

Dans cette expérimentation nous avons effectué des opérations d'accès aux données (lecture/écritures) en attribuant des poids de différentes manières (automatique, manuelle et aléatoire).

Le tableau 5 présente les temps de réponse moyens par rapport aux nombres de copies. Pour des opérations de lectures.

Tableau 5 – Temps de réponses moyens par nombre de copies pour lecture

	Nombre de copies	10	15	20	25	Moyenne
Temps de réponse moyens (ms)	**Automatique**	550	591.95	565.9	645.45	588.325
	Manuel	570.15	542.95	549.9	614.75	569.4375
	Aléatoire	542.45	638.9	706.75	722.85	652.7375

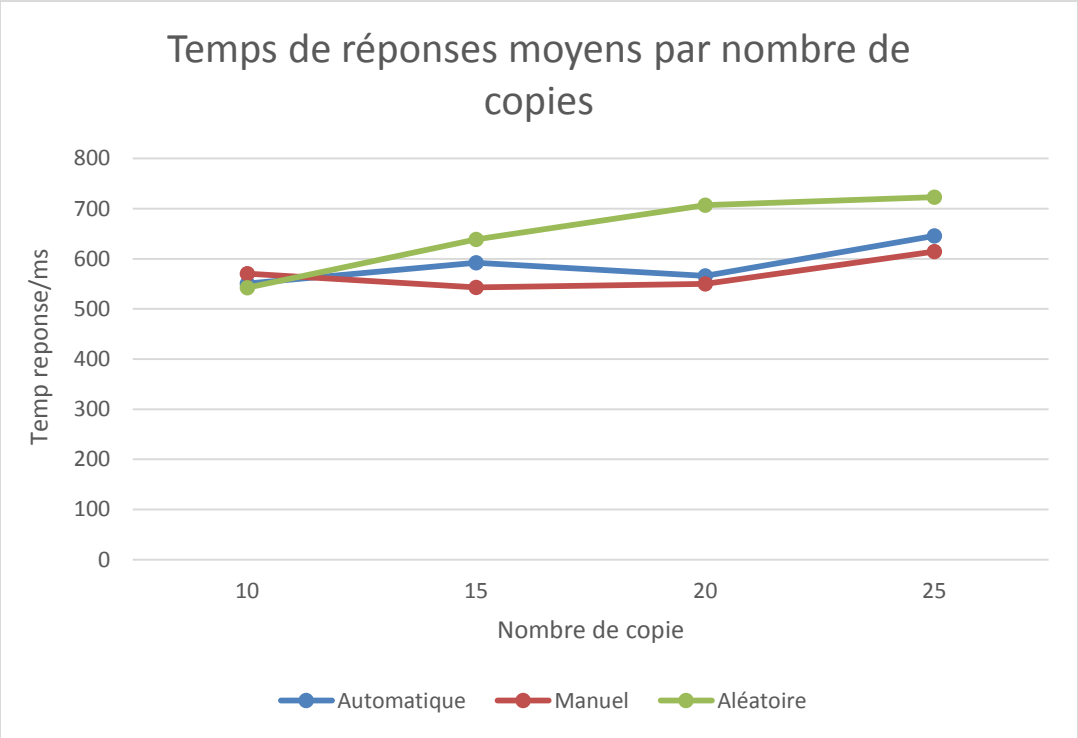


Figure 22 – Temps de réponses moyens par nombre de copies pour lecture

Les résultats obtenus dans le tableau 3 sont modélisés par l'histogramme de la figure 22 ci-dessus, qui représente les temps de réponses moyens par rapport les fonctionnalités (lecture).

Nous remarquons que la fonctionnalité (manières) manuelle est la meilleure parce que les temps de réponses moyens sont les plus faibles.

Le tableau 5 présente les temps de réponse moyens par rapport aux nombres de copies. Pour des opérations de écritures.

Tableau 6 – Temps de réponse moyens par nombre de copies pour l'écriture

	Nombre de copies	10	15	20	25	Moyenne
Temps de réponse moyens (ms)	**Automatique**	556	564.35	570.4	576.55	566.825
	Manuel	569.95	566.35	557.4	573.35	566.7625
	Aléatoire	582.95	570.55	579.4	589	580.475

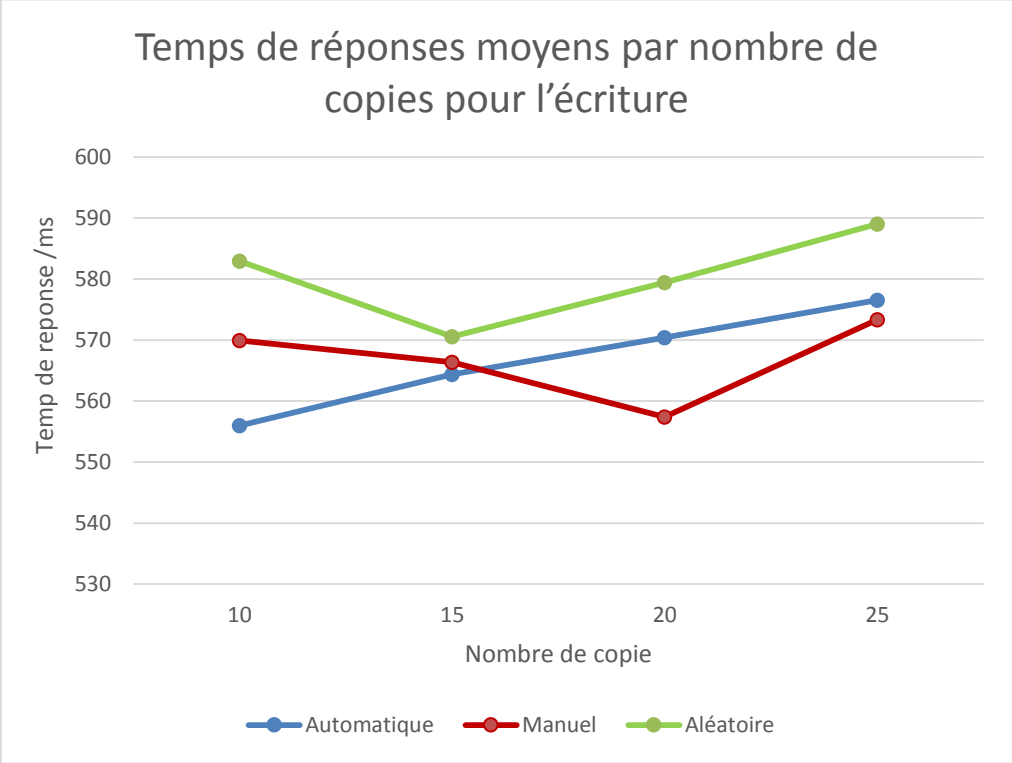


Figure 23 – Temps de réponses moyens par nombre de copies pour l'écriture

Les résultats obtenus dans le tableau 4 sont modélisés par l'histogramme de la figure 23 ci-dessus, qui représente les temps de réponses moyens par nombre de copies pour l'écriture.

Nous remarquons que la fonctionnalité (manières) manuelle est la meilleure parce que les temps de réponses moyens sont les plus faibles.

4.5 Conclusion

Dans ce chapitre, nous avons décrit en détail notre application. Nous avons ensuite présenté les différents résultats des expérimentations menées par notre application pour évaluer notre approche.

Dans les votes pondérés selon les expérimentations à notre niveau, le paramètre processeur est le meilleur par rapport aux les différents paramètres et la meilleure d'attribution est la méthode manuelle.

.

Conclusion Générale

Les SDGE sont utilisés pour les environnements de développement et d'exécution des applications à grande échelle, mais ils sont des systèmes instables où les nœuds sont susceptibles de se déconnecter du système à tout moment. Plus particulièrement lors de la déconnexion d'un nœud, ces données peuvent disparaitre à leur tour provoquant ainsi leur perte.

Parmi les solutions proposées pour répondre au problème de perte des données nous avons cité la réplication des données qui permet de fournir une meilleure tolérance aux pannes et le passage à l'échelle. En effet, elle permet d'améliorer la fiabilité et les performances du système, en réduisant les coûts d'exécution des applications.

La disponibilité des données peut assurer l'accès aux données répliquées même en cas de défaillance d'un nœud. Bien que très avantageuse, la réplication présente un inconvénient de cohérence des données où toutes les copies d'une même donnée doivent être théoriquement identiques. En effet, lorsqu'une copie est modifiée dans les SDGE, la modification doit être propagée à toutes les autres copies du système afin de garantir leur cohérence. Cependant, le maintien de la cohérence de toutes les répliques représente un défi majeur dans un tel environnement où le passage à l'échelle est important. Il est donc nécessaire de faire une étude pour répondre à ces problèmes de cohérence.

Le but de notre mémoire était de présenter une approche sur l'utilisation des votes pondérés dans la gestion de la cohérence des données dans les SDGE.

D'après les résultats obtenus par les différentes expérimentations, nous avons constaté que notre approche de vote pondéré permet la réduction du temps de réponse comme nous avons vu parce que les nombres des nœuds qui ont des plus grands poids été les plus utilisé.

Comme perspectives pour les futurs travaux, nous envisageons d'appliquer et d'étudier notre méthode basée sur les votes pondérés dans un environnement plus réaliste. Nous projetons aussi de comparer notre approche avec d'autres approches de gestion de la cohérence dans les SDGE.

Bibliographie

Rapport Technique

[1] Belkhir, M., Benaouda, M. : Modélisation UML et mise en place des indicateurs d'analyse spatio-multidimentionnelle dans un SOLAP pour la gestion de sécurité dans la ville de Mostaganem. Faculté des sciences et exactes et d'informatique, Département Mathématiques et informatique, Université de Abdelhamid Ibn, Mostaganem, 2014-2015.

[2] Bouharaoua, F. : Stratégies de réplication dynamique dans les grilles de données. Faculté des sciences et sciences de l'ingénieur, Département d'informatique, Université de Abdelhamid Ibn, Mostaganem, juin 2007.

[3] Dakhli, A., Matar, B. : Conception et Développement d'une application de gestion d'une base de connaissances au sein de la CNAM. Université virtuelle de Tunis, Tunis, 2009.

[4] El Morhder, Y. : La gestion de la qualité. Faculté polydisciplinaire. Université de Chouaib Doukali d'El Jadida, Maroc, 2008.

[5] Hadi, N. : Grille de calcul. Université d'oran1, Oran, mars 2020.

[6] Hadi, N. : Réplication et ordonnancement dans les grilles de calculs. Université d'oran1, Faculté des Sciences, Département d'informatique, Oran, 2013.

[7] Kateb Hachemi Amar, A. : Cohérence des données répliquées sur les grilles. Département d'informatique, Université des sciences et de la technologie Mohamed Boudiaf, Oran, 2012 - 2013.

[8] Kouidri, S : Gestion de la cohérence des répliques tolérante aux fautes dans les grille de données. Faculté des Sciences, Département d'informatique , Oran, 2011.

[9] Moghrani, A., Azzoug , Z. : Conception et réalisation d'une application de suivi de patients dans un établissement hospitalier. Université abederrahmane mira-bejaia, Bejaia, 2016-2017.

[10] Tshiamua, J. : Mise en oeuvre d'un système distribué pour l'identification et le suivi du casier judiciaire. Université pédagogique nationale, 2015-2016.

Article

[11] Chiky, E., Kumar, S., Lefebvre, S., Gressier Soudan, E. : *Protocole de gestion de la cohérence dans les systèmesde stockage distribués*, 75006 Paris, France, 2016, pp 207-208.

[12] Mason, T. : *Weighted Voting for Replicated Data, Distributed Systems, mars 2018, page 5.*

[13] Zakaria, M. : *Une introduction aux systémes distribués*, Tech Blog Ingeniance, juillet 2020, page 1.

Livre

[14] Pankouski, T : A Consensus Quorum Algorithm for Replicated NoSQL Data. Poznan University of Technology, page 117, Pologne, 2015.

Documents web

[15] Java, https://www.java.com/fr/download/help/whatis_java.html.

[16] SlidePlayer, https://slideplayer.fr/slide/4205514.

[17] Wikipedia.org, https://fr.wikipedia.org/wiki/NetBeans.