THÈSE

En vue de l'obtention du

DOCTORAT DE L'UNIVERSITÉ DE TOULOUSE

Délivré par : *l'Université Toulouse 3 Paul Sabatier (UT3 Paul Sabatier)*

Présentée et soutenue le *04/05/2022* **par :**

Morgan SÉGUÉLA

Stratégie de réplication de données prenant en compte la consommation énergétique et la dépense dans les systèmes à grandes échelles

JURY

Nouredine MELAB	Professeur des Universités, Université de Lille	Rapporteur
Sébastien MONNET	Professeur des Universités, Université Savoie Mont Blanc	Rapporteur
Francine KRIEF	Professeure des Universités, INP Bordeaux	Examinatrice
Esther PACITTI	Professeure des Universités, Université de Montpellier	Examinatrice
Patricia STOLF	Maîtresse de Conférences HDR, Université Toulouse Jean Jaurès	Examinatrice
Jean-Marc PIERSON	Professeur des Universités, Université Toulouse 3	Directeur de Thèse
Riad MOKADEM	Maître de Conférences HDR, Université Toulouse 3	Co-Directeur de Thèse

École doctorale et spécialité :
MITT : Domaine STIC : Réseaux, Télécoms, Systèmes et Architecture
Unité de Recherche :
Institut de Recherche en Informatique de Toulouse (UMR 5505)
Directeur(s) de Thèse :
Professeur Jean-Marc PIERSON et *Maître de Conférences HDR Riad MOKADEM*
Rapporteurs :
Professeur Sébastien MONNET et *Professeur Nouredine MELAB*

Remerciements

Je souhaite avant tout exprimer mes sincères remerciements au Professeur Jean-Marc Pierson (Directeur de thèse) et au Maître de Conférence Riad Mokadem (Co-directeur de thèse), pour avoir dirigé mes travaux de thèse, pour leur confiance, leurs conseils et soutien tout au long de cette thèse.

Je remercie très sincèrement les rapporteurs de ma thèse, M. Nouredine Melab, Professeur à l'Université de Lille, et M. Sébastien Monnet, Professeur à l'Université de Savoie Mont Blanc, qui m'ont fait l'honneur d'apporter leur précieuse expertise dans l'évaluation de mes travaux.

Je souhaite aussi exprimer mes remerciements à Mme Francine Krief, Professeure à l'INP Bordeaux ; Mme Esther Pacitti, Professeure à l'Université de Montpellier ; et Mme Patricia Stolf, Maîtresse de conférence à l'Université Toulouse Jean Jaurès d'avoir accepté d'examiner mes travaux de recherche.

Je remercie aussi les autres membres des équipes qui m'ont accueilli, les Professeurs Franck Morvan et Abdelkader Hameurlain et la Maitresse de Conférence Shaoyi Yin pour l'équipe Pyramide. Et je remercie les Maitres de Conférences Paul Renaud-Goud, Georges Da Costa et Amal Sayah pour leur accueil au sein de SEPIA.

Mes remerciements vont aussi à la direction de l'IRIT de m'avoir accueilli et pour le bon déroulement de la thèse.

Je souhaite évidemment remercier l'ensemble de mes collègues de l'IRIT Tanissia Djemai, Florent Dubois, Léo Grange, Gustavo Rustirolla, Meryem Zaid au sein de l'équipe SÉPIA, Antoine Bugnicourt, Mehdi Kandi et Damien Wojtowicz dans l'équipe Pyramide, et enfin Zineb Benis, Nicolas Bizzozzero, Lila Boualili, Olivier De Casanove, Malik Irain, Luis Lugo, Paul Mousset, Gia-Hung Nguyen, Raphaêl Sourty. Je voulais aussi et surtout remercier Julie Budaher qui m'a beaucoup aidé et soutenu durant cette fin de thèse.

Un grand merci aussi à mon entourage et amis de Toulouse, de France et de par le monde pour leur soutien.

Enfin, je souhaite remercier de tout mon cœur ma mère et ma famille pour leur enthousiasme, leur soutien et leur patience durant ces dernières années.

Résumé

Les applications d'aujourd'hui nécessitent l'accès à des données réparties à travers le monde. L'augmentation du volume de ces données conduit à des problématiques de disponibilité et de performance, surtout lorsque ces données sont fréquemment requêtées. Une manière de répondre à ces problèmes est la réplication de données, une technique très utilisée dans les systèmes distribués classiques, mais également dans les systèmes à grande échelle. De nombreuses stratégies de réplication de données ont été proposées dans de tels systèmes. Elles visent à déterminer les données à répliquer, combien de répliques créer, quand créer les répliques et où les placer. Dans ce manuscrit, nous proposons une stratégie de réplication de données dans les systèmes Cloud.

De nos jours, les enjeux environnementaux deviennent des problématiques majeures dans notre société. Certaines entreprises cherchent à avoir un impact positif sur ces enjeux. Pour les fournisseurs de services numériques et de Cloud, cela se traduit par une réduction des émissions de gaz à effet de serre en réduisant la consommation énergétique et en utilisant des ressources énergétiques plus vertes.

Dans le cadre de cette thèse, nous proposons une stratégie de réplication de données qui prend en compte ces problématiques économiques et énergétiques. Un placement initial, statique, est mis en place pour répondre aux objectifs de disponibilité et de tolérance aux fautes tout en tenant compte de la consommation énergétique et de la dépense du fournisseur. Ce placement s'appuie sur l'hétérogénéité entre les centres de données et sur l'utilisation de technique de veille permettant de réduire la consommation énergétique de serveurs inutilisés. Un tel placement permet de choisir la politique à mettre en place selon l'intérêt du fournisseur.

Une gestion dynamique des répliques, s'appuyant sur le placement initial, est ensuite proposée. Elle permet de s'adapter aux variations de la charge de travail, détectée à l'aide de Cartes de contrôle. Ces cartes s'appuient sur des probabilités pour lever des alertes. D'un côté, de nouvelles répliques sont créées lorsque la charge augmente, tout en tenant compte de la consommation énergétique. D'un autre côté, des répliques sont supprimées lorsque cette charge diminue. Cela permet de réduire les coûts et la consommation énergétique liés au stockage.

Une évaluation de performances, via une simulation, a permis de valider la stratégie proposée tout en comparant ses performances à celles d'autres stratégies proposées dans la littérature. Pour le placement initial, ces évaluations ont permis de mettre en avant l'impact des différents choix proposés, mais aussi de montrer qu'un placement initial des répliques plus "intelligent" peut avoir un impact positif sur les performances et la disponibilité, tout en réduisant la consommation énergétique et les dépenses. Puis, nous l'avons évalué en intégrant la gestion dynamique des répliques. Les résultats de la partie dynamique montrent qui nous arrivons à améliorer les performances par rapport au placement initial, tout en réduisant à la fois les dépenses et la consommation énergétique. De plus, cela permet de mettre en avant l'importance des choix et des méthodes mises en place lors de l'ajout ou de la suppression des répliques.

Mots-clés : Système Cloud, Réplication de données, Modèle de coût économique, Consommation énergétique, Service Level Agreement

Abstract

Nowadays, applications need access to data across the world. And the volume growth of these data leads to availability and performance issues, especially when these data are heavily requested. A way to answer issues is data replication, a commonly used technique in distributed systems, but also in large scale ones. Many data replication strategies have been proposed for these kinds of systems. They seek to choose which data to replicate, how many replicas, when and where to replicate them. In this thesis, we propose a data replication strategy for the Cloud systems.

Nowadays, environmental issues are becoming more and more important in our society. Several companies seek to have positive impacts on these issues. For IT companies and Cloud providers, these issues are being answered reducing carbon footprint by reducing their energy consumption and the use of greener energy resources.

In the context of this thesis, we propose a data replication strategy that considers both energetic and economic issues. It starts with an initial placement to answer availability and fault tolerance issues while taking into account energy consumption and expenditure. This placement uses heterogeneity between data centers and the use of technologies that puts server into sleep mode to reduce energy consumption of unused servers. This placement permits to let the administrator choose the policy they want to apply according to the provider's interest.

A dynamic replica management, supported by the initial placement, is then studied which permits to adapt to workload variations. These workload variations are detected through Control Charts, which uses probabilities to rise warnings. When the workload increases, the proposed data replication will create new replicas while taking into account energy consumption. Then, when the workload decreases, the strategy will delete replicas to reduce storage energy consumption and expenditure.

Performance evaluations, through simulation, permits to validate the proposed data replication strategy while comparing its performances with other strategies proposed in the literature. For the initial placement, these evaluations permits to highlight the impact of different suggested choices, and also show the positive impact of an "intelligent" initial placement on availability and performance, while reducing energy consumption and expenditure. Then we added the dynamic data management to the evaluations. Results show that we achieve to have better performance compared to the initial placement, while keeping reducing energy consumption and expenditure. It also highlights the importance of choices and methods used when creating or deleting replicas.

Keywords : Cloud, Data replication, Expenditure models, Energy consumption, Service Level Agreement

Table des matières

Table des figures

Liste des tableaux

Chapitre 1

Introduction

Sommaire

1.1 Contexte

La quantité de données produites et utilisées dans le monde ne cesse d'augmenter. Cette augmentation peut être illustrée par la quantité de données stockées dans le monde. Les rapports annuels de Seagate, producteur de disques durs, présentent le volume de stockage vendu par trimestre dans les informations supplémentaires [Larg, 2020, Larg, 2021]. Ces rapports montrent, que lors du dernier trimestre d'activité de 2019, un total de 84.5 ExaBytes [1] de stockage de données sur disque ont été vendu. Ce chiffre atteint 123.3 EB lors du dernier trimestre de 2021.

De plus, Seagate et IDC ont fourni un rapport qui évalue le volume de données stockées dans le monde [Dave, 2021]. Cette évaluation montre que la quantité de données stockées en 2016 était de 20 ZetaByte [2], puis 41 ZB en 2019 et ils estiment ce volume à 80 ZB en 2022.

Une autre illustration de cette augmentation de la quantité de données produites et utilisées se retrouve dans l'article [Morley et al., 2018]. Les autrices présentent une augmentation de la quantité de données transférées dans le monde qui passe de 100GB par seconde en 2001, à 26,000GB par seconde en 2016. Une grande partie de la consommation du réseaux provient du divertissement temps réel tel que YouTube ou les plateformes de streaming par exemple.

Cette augmentation de la quantité de données implique des problématiques de disponibilité et de performance. En effet, ces différentes données peuvent être accédées par différentes entités, comme les entreprises ou les particuliers, pour différentes raisons, que ce soit pour le divertissement, le débat, pour alimenter l'économie ou encore pour la production scientifique. Ces différentes données peuvent être accédées de n'importe où et à n'importe quel moment. Parfois certaines données sont massivement accédées durant une courte période de temps pouvant impliquer des problèmes de performance. Cela se traduirait par la création de goulots d'étranglement et une augmentation du temps de réponse.

Une manière de répondre à ces problématiques de disponibilité et performance est l'utilisation de la réplication de données. Dans ce contexte, de nombreuses stratégies de réplication de données ont été proposées pour des architectures qui ont évolué durant ces dernières années. Des architectures parallèles [Valduriez, 1993] au fog [Vales et al., 2019] en passant par les grilles de calcul [Lamehamedi

1. $1EB = 1000^3 GB$
2. $1ZB = 1000^4 GB$

et al., 2002] et le cloud [Tabet et al., 2017], la réplication de données répond à des objectifs tel que la disponibilité, la tolérance aux fautes et la réduction de la consommation de la bande-passante. Dans cette thèse, nous allons également exploiter la technique de la réplication de données en prenant en compte des problématiques telles que environnementales ou économiques par exemple.

De même, le *Cloud Computing* est une architecture avantageuse car elle permet de pouvoir manipuler une grande quantité de données à moindre coût . Les caractéristiques du Cloud sont présentées dans le chapitre 2. Mais, il est important de noter que les Cloud publics [Goyal, 2014], qui sont donc fournis et entretenus par des entreprises, permettent d'accéder à de la puissance de calcul sans investir dans du nouveau matériel. Les premières apparitions de Cloud publics remontent à 1990 avec l'entreprise Salesforce [Surbiryala and Rong, 2019]. Cependant, c'est en 2006 qu'est lancé Amazon Web Services, service de Cloud ayant connu depuis une explosion du nombre de clientes et clients et de leurs recettes. Cette explosion se voit d'autant plus ces dernières années avec les rapports annuels de Amazon [Bezos, 2017, Bezos, 2020] qui montrent que les recettes de AWS passent de 4.6 milliards de dollars en 2014 à 35 milliards de dollars en 2019. De même, les rapports de Microsoft [Nadella, 2018, Nadella, 2020] montrent que les recettes associées au cloud passent de 25 milliards de dollars en 2016 à 48.5 milliards de dollars en 2020.

Une caractéristique essentielle au Cloud est l'élasticité qui permet au fournisseur d'adapter les ressources de manière automatique aux besoins des locataires. Cette élasticité permet aux fournisseurs de mettre en place le modèle économique *Pay-as-you-go* [Kilcioglu et al., 2017] qui fait payer au locataire uniquement ce qu'il consomme comme ressources. Cependant, comme le décrit [Mubeen et al., 2018], il peut y avoir un écart entre les performances fournis par les fournisseurs ainsi que les prix associés et les besoins de la clientèle. Un contrat est donc signé entre chaque locataire et le fournisseur, nommé *Service Level Agreement* (SLA), qui permet de fixer les prix et les objectifs de niveau de service du fournisseur ainsi que les pénalités si ce dernier ne les atteint pas.

Cependant, ces problématiques économiques se lient assez peu d'un questionnement écologique. En effet, comme le montre [Belkhir and Elmeligi, 2018] l'industrie de l'information et de communication contribuait entre 1.06% et 1.6% aux émissions globales de gaz à effet de serre en 2007. Cette contribution aurait plus que doublé en 2020 avec une contribution entre 3.06% et 3.6% de l'émission de gaz à effet de serre global. Dans cette contribution, les centres de données représentent 33% de ces émissions en 2007 et 45% en 2020. De même, une étude à l'échelle de la France [Bordage et al., 2021] montre que cette industrie consomme 6.2% de la consommation énergétique primaire. Les centres de données sont responsables à hauteur de 4 à 15% de cet impact sur la consommation énergétique.

Un autre point de vue proposé par [Masanet et al., 2020] montre que malgré une augmentation de la charge de travail de 550% la consommation énergétique des centres de données n'a augmenté que de 6% grâce à une amélioration de l'efficacité énergétique. Cependant, cela reste important de réduire sa consommation énergétique et donc l'émission de gaz à effet de serre au vue des problématiques climatiques à venir.

1.2 Motivation

Deux types de motivations ont appuyé notre recherche durant cette thèse.

D'abord, nous nous sommes intéressés à la prise en compte des coûts économiques pour le fournisseur. Cela passe par le fait d'augmenter sa clientèle (en assurant une certaine qualité de service), et de profiter du réseau internet mondial pour ajuster son modèle économique entre les pays.

Nous nous sommes également intéressés à la prise en compte de la consommation énergétique du fournisseur lors de la réplication de données. En effet, il faut tenir compte des contraintes environnementales qui impliquent la réduction des émissions de gaz à effet de serre. Cette réduction d'émission peut aussi passer par une baisse de la consommation énergétique [Xu et al., 2015].

Ces deux problématiques ne sont cependant pas complètement indépendantes. En effet, du point de vue du fournisseur, elles se croisent à plusieurs moments. Par exemple, lors de l'investissement

pour du nouveau matériel pouvant être plus performant, permettant l'utilisation de techniques réduisant la consommation énergétique rendant ce matériel plus coûteux. De plus, la consommation énergétique a un impact sur les dépenses de plusieurs manières différentes, comme coût direct ($\pm$ 10% des dépenses) ou avec le marché du carbone. Enfin, de plus en plus d'utilisateurs et d'utilisatrices sont conscients de ces enjeux environnementaux et en tenir compte peut permettre d'attirer une nouvelle clientèle.

1.3 Questions Scientifiques

Dans cette thèse, nous proposons une stratégie de réplication de données dans le Cloud qui répond à des objectifs de disponibilité et de performance, tout en tenant compte des problématiques de consommation énergétique et de dépense en tant que fournisseur.

Dans un premier temps, avec la proposition d'un placement statique intégré dans une stratégie de réplication dynamique de données, une question se pose : Pourquoi faire du statique dans un environnement dynamique ?

Puis, dans le cadre de nos objectifs, il est important de répondre à la question : comment la consommation énergétique peut-elle être prise en compte dans la réplication de données ?

Comme le centre de la stratégie est de pouvoir prendre en compte à la fois la consommation énergétique et la dépense, la question qui se pose donc est : Comment permettre aux administrateurs et administratrices de moduler leurs choix selon leurs objectifs ?

Enfin, par rapport à la gestion dynamique des répliques, une question peut se poser sur la manière de mettre en place cette dynamicité, quels outils peuvent être adaptés aux déclenchements de la partie dynamique, sans aggraver l'impact énergétique tout en contrôlant l'incertitude ?

1.4 Contributions

Pour répondre aux questions présentées précédemment, nous proposons une stratégie de réplication de données. Les contributions de ce manuscrit peuvent être présentées de la manière suivante :

Placement Statique La réplication et le placement statique des répliques visent à réduire à la fois la consommation énergétique et la dépense. Pour cela, nous avons modélisé la consommation énergétique et la dépense de différentes actions appliquées aux données. Ces modèles sont utilisés dans un algorithme d'optimisation multi-objectifs, permettant de choisir quelle politique appliquer. Afin de s'adapter au contexte large échelle, le placement statique se fait en deux étapes. (i) Un placement inter-centres de données, qui choisit quels centres de données stockeront quelles répliques. Cette première étape s'appuie sur l'hétérogénéité entre centres pour tenir compte de la consommation énergétique et de la dépense. (ii) Un placement intra-centre de données, qui désigne sur quels serveurs de chaque centre de données seront stockées les répliques issues de la première étape. Dans ce placement, la stratégie va s'appuyer sur l'extinction des serveurs [Meisner et al., 2009] pour réduire la consommation énergétique

Placement dynamique Pour compléter la stratégie de réplication de données, une réplication de données dynamique est mise en place. Cette réplication a pour but d'adapter le nombre de répliques à l'augmentation et à la diminution de la charge de travail. Cette variation de charge est détectée à l'aide des cartes de contrôle [Shewhart, 1931]. Les cartes de contrôle sont un outil de contrôle statistique des procédés qui s'appuie sur les probabilités pour lever des alertes. Dans la stratégie proposée, les cartes s'appuient sur les violations de niveau de service, c'est-à-dire un temps de réponse supérieur aux objectifs signés dans le contrat, pour lever des alertes. Lorsque la carte de contrôle génère une alerte, la stratégie va créer et placer de nouvelles répliques ou en supprimer. L'ajout de nouvelles répliques nécessite un arbitrage entre les performances et la consommation énergétique. De même, la suppression de répliques, entraîne un arbitrage entre réduction de la

consommation énergétique et de la dépense dûe au stockage. Dans le cadre de cette suppression, les administrateurs et administratrices peuvent à nouveau influer sur l'objectif à prendre en compte.

1.5 Liste des publications

Le travail de thèse a donné lieu à deux articles scientifiques acceptés dans des conférences internationales :

— Morgan Séguéla, Riad Mokadem, Jean-Marc Pierson. Octobre 2019. *Comparing energy-aware and cost-aware data replication strategy.* Tenth International Green and Sustainable Computing Conference (IGSCC). Alexandria, VA, USA. IEEE.

— Morgan Séguéla, Riad Mokadem, Jean-Marc Pierson. Septembre 2021. *Energy and Expenditure Aware Data Replication Strategy.* IEEE 14th International Conference on Cloud Computing (CLOUD). Chicago, IL, USA. IEEE, p. 421-426.

— Morgan Séguéla, Riad Mokadem, Jean-Marc Pierson. À Paraître. *Dynamic Energy and Expenditure Aware Data Replication Strategy.* IEEE International Conference on Cloud Computing (CLOUD). Barcelone, Espagne.

Ainsi qu'un article accepté dans une conférence nationale :

— Morgan Séguéla, Riad Mokadem, Jean-Marc Pierson. Juin 2019. *Étude des Stratégies de réplication de données prenant en compte la consommation énergétique vs. le profit économique dans les systèmes Cloud.* Conférence d'informatique en Parallélisme, Architecture et Système (ComPAS 2019). Anglet, France.

1.6 Organisation du manuscrit

Ce manuscrit est constitué de sept chapitres. Après la description du contexte, les problématiques et les contributions présentées dans le chapitre 1, le reste du document est constitué comme suit :

Le chapitre 2 présente l'état de l'art. Nous positionnons nos travaux relatifs à la réplication de données dans les systèmes Cloud par rapport aux travaux existants dans la littérature. Nous présentons aussi les techniques de réduction de la consommation énergétique.

Le chapitre 3 introduit les modèles de consommation énergétique et de dépense utilisés tout au long de ce manuscrit. De plus, nous introduisons les outils qui seront utilisés dans la stratégie de réplication.

Le chapitre 4 est dédié à la description de l'architecture prise en compte lors de la proposition de notre stratégie de réplication de données et à l'environnement expérimental qui sera utilisé dans les chapitres suivants.

Le chapitre 5 décrit le placement initial de la stratégie de réplication de données. Elle est exécutée avant les premières requêtes utilisateurs et s'appuie sur un algorithme d'optimisation multi-objectif. Cette description sera suivi d'une évaluation des performances de ce placement initial.

Le chapitre 6 présente la réplication dynamique de la stratégie, qui vise à adapter le nombre et le placement des répliques à l'augmentation et à la baisse de charge. Pour cela, nous nous appuyons sur les cartes de contrôle afin de tenir compte de l'incertitude. Ce chapitre se terminera ainsi par une évaluation des performances de la stratégie de réplication.

Le chapitre 7 conclut le manuscrit de thèse avec une synthèse de la proposition de stratégie de réplication de données, suivi de différentes perspectives d'amélioration de la proposition.

Chapitre 2

État de l'art

Sommaire

2.1 Introduction

Dans ce chapitre, nous allons dans un premier temps introduire les principales caractéristiques du Cloud. Nous ferons ensuite une brève description de l'impact de la consommation énergétique des centres de données dans la consommation mondiale. Puis nous présenterons quelques techniques pour réduire la consommation énergétique de ces centres de données. Enfin, nous présenterons différentes stratégies de réplication de données en les classifiant selon leurs objectifs dans un premier temps, puis selon qu'elles sont statiques ou dynamiques. Nous conclurons ce chapitre, en mettant en avant les aspects positifs et négatifs des stratégies de réplications de données et pour positionner nos travaux par rapport à ces derniers.

2.2 Environnement Cloud

Dans cette partie, nous allons décrire plus en détails les caractéristiques du Cloud, ainsi que quelques enjeux associés à cette architecture. De plus, nous introduirons les problématiques énergétiques et les solutions proposées à appliquer dans les centres de données. Cela permettra de mettre en avant les outils sur lesquels s'appuie la stratégie de réplication de données.

2.2.1 Caractéristiques

Définition : Nous allons dans un premier temps définir l'architecture du Cloud en se basant sur [Buyya et al., 2009] (traduit de l'anglais) qui définit le Cloud comme :

"Un système de type parallèle et distribué qui consiste en une collection d'ordinateurs virtualisés inter-connectés qui s'approvisionnent dynamiquement et présentés comme une ou plusieurs ressources de calcul unifiées basées sur un contrat de niveau de service établi sur une négociation entre le fournisseur de service et l'usager"

L'élasticité : Le Cloud se base sur de la virtualisation, qui permet d'approvisionner dynamiquement les resources des machines virtuelles. Cet approvisionnement est fait de manière élastique comme le définit [Herbst et al., 2013]. Cela signifie que l'approvisionnement se fait de manière automatique et adapte les ressources au plus proche des besoins de chaque locataire. Le système évitera tout problème de sous-provisionnement, ou de sur-provisionnement. Cette élasticité permet de mettre en place un modèle économique nommé *Pay-as-you-go* [Armbrust et al., 2010] qui implique que le locataire paie uniquement ce qu'il consomme.

Contrat de niveau de service : Le prix de la location des ressources est explicité dans le contrat de niveau de service ou SLA (*Service Level Agreement*) signé entre le fournisseur et le locataire. Ce contrat contient, en plus des coûts de location, les objectifs de niveau de service que le fournisseur doit atteindre et les pénalités si ces objectifs ne sont pas atteints.

Services disponibles : Les services fournis sont de différents niveaux d'abstraction selon les besoins de l'utilisateur et de l'utilisatrice [Goyal, 2014]. Le premier niveau est l'*Infrastructure as a Service* (IaaS), fournissant une machine virtuelle qui inclue uniquement les bases : du stockage, de la puissance de calcul et de la capacité réseaux. Cela correspond à AWS EC2 par exemple. Le second niveau est la *Plateform as a Service* (PaaS) qui en plus de l'IaaS ajoute des outils de base de développement et de déploiement. Dans le cadre d'Amazon, le PaaS correspondrait à AWS Elastic Beanstalk. Enfin le dernier niveau est le *Software as a Service* (SaaS), il fournit uniquement un logiciel qui tourne sur le Cloud. Un exemple de SaaS est l'outil d'édition de texte Google Docs.

Modèles : Différents modèles de Cloud sont mis en place en fonction des objectifs [Goyal, 2014]. Les Clouds privés sont mis en place au sein de grosses entreprises pour permettre de rendre disponibles de la puissance de calcul, mais cela implique un investissement initial. À l'inverse, les Clouds publics sont mis en place par des entreprises qui vendent leurs puissances de calcul à d'autres ou des particuliers comme AWS ou Microsoft Azure. Il existe aussi, des modèles hybrides qui consistent en l'utilisation d'un cloud privé qui est étendu avec un cloud public en fonction des besoins de l'entreprise.

Un résumé des points précédents est présenté par la *National Institute of Standard and Technologies* [Mell and Grance, 2011].

Les Clouds publics présentés précédemment se sont de plus en plus démocratisés car ils permettent à des utilisateurs et utilisatrices d'accéder à de la puissance de calcul à un moindre coût. Comme le montre les résultats d'Amazon, la vente nette d'Amazon Web Services est passée d'environ 12 milliards de dollars en 2016 [Bezos, 2017] à 35 milliards en 2019 [Bezos, 2020]. Cependant, cette augmentation de la clientèle et de ressources nécessaires implique une augmentation du nombre de

centres de données pour répondre à ces besoins. Cela a un impact sur la consommation énergétique globale des centres de données au niveau mondial.

2.2.2 Réduire la consommation énergétique

2.2.2.1 Consommation énergétique

Les auteurs [Koomey, 2011] montrent qu'en 2010 la consommation énergétique des centres de données à travers le monde représente entre 1.1% et 1.5% de la consommation énergétique mondiale. Un indicateur pour estimer l'efficacité énergétique d'un centre de données est le *Power Usage Effectiveness* (PUE).

$$PUE = P(\text{Centre de données})/P(\text{Matériel Informatique}) \tag{2.1}$$

Les auteurs estiment que le PUE moyen en 2010 était entre 1.92 et 1.83 avec un minimum de 1.36 et un maximum de 3.6. Un article plus récent [Lloyd and Rebow, 2018] estime qu'en Europe le PUE est de 1.8 en moyenne, et de 1.89 aux États-Unis avec des sites qui peuvent être entre 1.4 et 1.1. Cette baisse de PUE a eu un impact positif sur la consommation énergétique des centres de données. En effet, les auteurs de [Masanet et al., 2020] montrent que malgré une augmentation de 550% des besoins de calcul entre 2010 et 2018, il y a eu une augmentation de la consommation énergétique de 6%. Cette amélioration de l'efficacité montre l'importance des outils et techniques de réduction de consommation énergétique. Avec l'augmentation de la quantité de données stockées et manipulées, la consommation énergétique nécessaire continuera d'augmenter. Et l'amélioration des techniques ainsi que leurs meilleures prises en compte sont nécessaires pour tenir compte des problématiques environnementales futures.

2.2.2.2 Efficacité énergétique

Pour améliorer l'efficacité énergétique des centres de données plusieurs techniques sont utilisées. Plusieurs études ont été faites pour répertorier ces techniques [Orgerie et al., 2014, Mastelic et al., 2014, Hamzaoui et al., 2020]. Ces études proposent des taxonomies pour classer ces techniques, les auteurs et autrice de [Orgerie et al., 2014] proposent une taxonomie des techniques pour réduire la consommation énergétique des nœuds de calcul. Cette dernière est traduite et présentée dans la figure 2.1.

Dans cette figure, nous avons mis en évidence les techniques sur lesquelles nous nous appuyons. La première technique utilisée est la **modélisation** de la consommation énergétique qui fait partie de la décision de réplication et de placement de données, et de la comparaison des stratégies de réplication de données. Cette modélisation sera présentée dans le chapitre 3.

Le contexte du Cloud permet l'utilisation de la **virtualisation**, elle a l'avantage de pouvoir isoler les performances et d'approvisionner la machine virtuelle en ressource selon les tâches à effectuer. Cet approvisionnement permet d'augmenter ou de réduire les ressources, donc d'allumer et d'éteindre des nœuds.

Dans le cadre de notre travail, nous nous sommes concentrés sur la **consolidation**, en essayant de placer les données sur une petite quantité de nœuds pour regrouper la charge de travail. Cette technique est couplée avec des niveaux de veille, l'idée est de mettre dans des **états de veille** les nœuds qui sont inactifs afin de réduire la consommation énergétique. Il faut cependant tenir compte du fait que le retour à l'activité normale d'un nœud implique un pic de puissance. Ainsi, si l'utilisation de cette technique n'est pas couplée avec un algorithme d'ordonnancement de tâches qui prend cette technique en compte, il est alors possible qu'un nœud augmente sa consommation énergétique en enchaînant les extinctions et allumages. Comme nous le verrons par la suite ces techniques sont utilisées dans le cadre de la réplication de données ayant pour objectif de réduire la consommation énergétique.

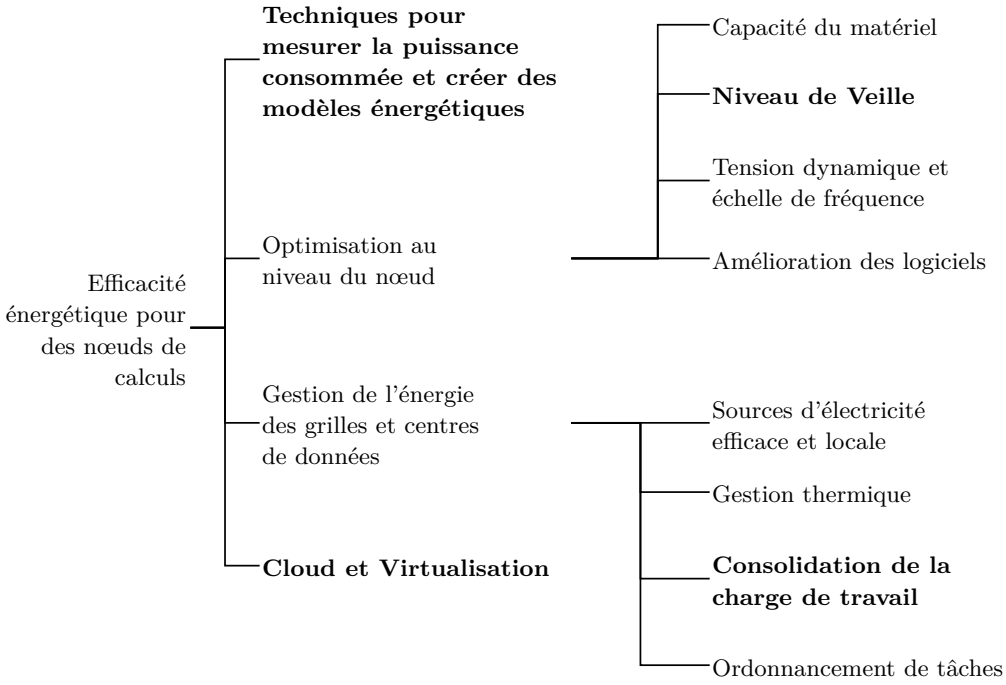


FIGURE 2.1 – Vue d'ensemble des techniques pour améliorer l'efficacité des nœuds de calcul [Orgerie et al., 2014] (traduit de l'anglais)

2.3 Réplication de données

Nous allons à présent faire un état de l'art des stratégies de réplication de données. Cet état de l'art se concentrera principalement sur les stratégies qui s'appuient sur une architecture Cloud, et mettra en avant celles qui tiennent compte des problématiques énergétiques ou économiques.

2.3.1 Introduction

La réplication de données est une technique permettant de répondre à des objectifs de disponibilité et de performance. Cette technique est étudiée depuis plusieurs décennies selon les architectures sur lesquelles la réplication s'applique, que ce soit des architectures parallèles [Valduriez, 1993], les grilles de calcul [Lamehamedi et al., 2002], plus récemment sur des architectures Cloud [Tabet et al., 2017] et fog [Vales et al., 2019]. Ainsi, des stratégies de réplication de données sont mises en place pour tenir compte de divers objectifs comme nous le verrons par la suite, en fonction des architectures sur lesquelles ces stratégies sont appliquées. Les stratégies de réplication doivent répondre à 4 questions [Tos et al., 2017] :

1. Quelle donnée répliquer ?
2. Quand répliquer ?
3. Combien de répliques créer ?
4. Où répliquer ?

Certaines stratégies de réplication de données se concentrent sur les problèmes de consistance des données lorsque celles-ci sont modifiées telle que la stratégie proposée par [Mauffret et al., 2019]. Dans le cadre de nos recherches, nous nous concentrons avant tout sur les stratégies de réplication de données en tenant compte des tâches de type *Online Analytical Processing* (OLAP) qui sont surtout de la lecture intensive et peu d'écritures.

2.3.2 Classification par objectif

Comme cela a été présenté dans l'introduction, la réplication de données est une technique ayant comme objectifs d'améliorer la disponibilité des données, et les performances. Pour cela les stratégies de réplication de données, répondent de manières différentes selon leurs objectifs, mais aussi selon les problèmes qu'elles prennent en compte. En effet, certaines stratégies prennent en compte les problématiques associées à l'énergie quand d'autres prennent en compte les problématiques économiques. Nous allons donc dans un premier temps présenter une classification par objectif des stratégies, puis une classification selon les problèmes qu'ils prennent en compte dans leurs solutions.

2.3.2.1 Disponibilité

Tout d'abord, nous étudions des stratégies qui tiennent compte principalement de la disponibilité.

Les auteurs de [Wei et al., 2010] proposent une stratégie de réplication de données qui s'appuie sur la probabilité de panne d'un nœud pour estimer un nombre de répliques minimum afin d'atteindre le niveau de disponibilité attendu. De plus, cette stratégie s'appuie sur le découpage d'une donnée en bloc afin de placer ces blocs de manière à équilibrer la charge de travail. L'idée ici est donc de minimiser le coût du stockage, mais dans ce cadre les coûts ne sont pas modélisés.

Une autre stratégie qui tient compte de la disponibilité proposée par [Pamies-Juarez et al., 2011] se base sur l'hétérogénéité entre les nœuds d'un Cloud pour tenir compte de la probabilité de non disponibilité d'un fichier selon le nœud sur lequel la donnée sera requêtée.

Un dernier exemple de stratégie qui met en avant les problématiques de disponibilité est [Selvi and Anbuselvi, 2018]. Cette stratégie s'appuie sur la fréquence d'accès des données, pour augmenter leur disponibilité si elles sont trop requêtées et la réduire si elles sont trop peu requêtées. Ainsi, les données trop peu requêtées sont réduites à 2 répliques.

D'autres stratégies qui ont pour objectif principal la disponibilité, prennent en compte des problématiques de coût et de consommation énergétique. Les auteurs de [Li et al., 2011] considèrent un objectif de disponibilité en se basant sur la probabilité de pannes et le coût de stockage. Cette stratégie sera détaillée un peu plus tard dans l'état de l'art.

2.3.2.2 Performance

D'autres stratégies proposées s'intéressent aux problématiques de performances.

Les auteurs de la stratégie de réplication proposée dans [Qu and Xiong, 2012] s'intéressent aux problématiques de disponibilité, de performance, tout en réduisant le nombre de répliques et de migrations. Pour cela, elle met en place un partitionnement des fichiers en blocs et un système d'acheminement des blocs pour répondre plus rapidement aux requêtes. Le système d'acheminement permet aussi de contrôler la charge des nœuds qui servent à l'acheminement des blocs. Lorsque cela n'est plus suffisant, les nœuds mettent en place de manière décentralisée une migration des répliques pour mieux équilibrer la charge ou la réplication pour rapprocher les données des points de tension. Enfin, un mécanisme de suppression de réplique est mis en place lorsque la charge diminue, tout en gardant la disponibilité au même niveau assurée par un nombre de répliques minimum calculé à partir de la probabilité de panne d'un nœud.

Une autre stratégie proposée par [Kumar et al., 2014] met aussi en place un partitionnement des données en minimisant le transfert de résultats intermédiaires lors de l'exécution d'une requête. Pour répondre à cette problématique, la stratégie construit un graphe au fur et à mesure des requêtes qui regroupe les fragments qui ont été requêtés ensemble. Lorsqu'un fragment est trop accédé, car faisant partie de différents sous-ensembles de fragments requêtés, il est répliqué pour repartir ces sous-ensembles sur chaque réplique. De plus, cette stratégie propose plusieurs techniques pour répondre à différents problèmes comme la prise en compte de la disponibilité ou son passage à l'échelle. Il est intéressant de noter que malgré l'absence de prise en compte de la consommation énergétique dans ses objectifs, les auteurs ont remarqué que leur stratégie avait un impact positif sur la diminution de la consommation énergétique.

Dans [Mansouri et al., 2017], les autrices proposent une stratégie de réplication de données basée sur la popularité des données. Pour cela, à l'aide du nombre de fois qu'une donnée est accédée, la stratégie calcule un score de popularité et réplique les données les plus populaires. Le choix du nœud sur lequel la donnée sera répliquée se fait selon un score de mérite qui base ses critères sur son espace disponible, sur sa proximité avec d'autres nœuds et la popularité du fichier à répliquer sur ce nœud. Comme dans les stratégies présentées précédemment, un découpage des données en fragments est mis en place, dont la taille de chaque fragment diffère selon le score de mérite du nœud qui le stockera.

[Bai et al., 2013] introduit une stratégie dynamique de réplication de données qui s'intéresse aussi aux problématiques de performance en se basant sur le temps de réponse autant pour choisir la donnée à répliquer que pour choisir le nœud qui stockera les répliques. Cette stratégie essaie de minimiser le nombre de répliques créées pour un temps de réponse maximal donné en utilisant un algorithme glouton. De plus, elle propose une méthode de placement de tâches pour prendre en compte les capacités de calcul du processeur, du réseau et d'entrée/sortie du disque.

Enfin, les auteurs de [Sousa et al., 2018] s'intéressent aussi à la performance du point de vue du temps de réponse, mais en tenant compte des objectifs de niveau de service de chaque utilisateur et utilisatrice. En effet, sur la base de différentes informations en entrées telles que le temps de réponse, la taille des données, la stratégie estime la durée jusqu'à la prochaine violation à l'aide de modèles linéaires afin de répliquer avant que la violation arrive. De même, un mécanisme de suppression de répliques est mis en place lorsque le temps de réponse diminue.

Dans les stratégies qui tiennent compte des problématiques de dépense et de consommation énergétique, une grande partie considèrent les performances comme objectif principal avec une considération de disponibilité.

Ainsi, [Janpet and Wen, 2013,Mansouri and Buyya, 2019,Hsu and Kshemkalyani, 2019] cherchent à minimiser le temps de réponse tout en réduisant les dépenses. De même, les stratégies [Tos et al., 2017, Mokadem and Hameurlain, 2020] utilisent les violations de niveaux de services comme indicateur de performance tout en tenant compte des problématiques économiques. Ces stratégies sont décrites dans la section 2.3.3.2

Enfin, les stratégies qui tiennent compte de la consommation énergétique ont tendance à considérer les performances du point de vue du nombre d'accès [Boru et al., 2015, Zhang et al., 2015, Li et al., 2019]. Cela permet de classer les données en fonction de leur chaleur, c'est-à-dire de leur fréquence d'accès, et donc d'éviter les goulots d'étranglement. Ces stratégies seront plus précisément décrites dans la section 2.3.3.1.

2.3.2.3 Analyse

Un résumé des objectifs pris en compte est présenté dans le tableau 2.1 avec les problématiques associées.

Répondre aux objectifs de disponibilité se fait au travers de 2 moyens. Soit (i) la stratégie tient compte de la probabilité de panne d'un nœud en la modélisant afin de garder au long terme un objectif de disponibilité [Wei et al., 2010, Li et al., 2011, Long et al., 2014]. Soit (ii) la stratégie va créer un nombre minimum de répliques en s'appuyant sur ces probabilités de pannes. Cette valeur minimum peut être définie par les usagers ou par les administrateurs et administratrices afin de répondre à un objectif de niveau de service de disponibilité ou en se basant sur de précédent travaux [Qu and Xiong, 2012, Janpet and Wen, 2013].

De même, une grande partie des stratégies s'intéressent aux performances de manière très différentes. Dans un premier temps, la majorité des stratégies qui ne prennent pas en compte la consommation énergétique ou la dépense découpent les données en fragments pour améliorer le temps de réponse en récupérant les fragments de manière parallèles [Wei et al., 2010,Kumar et al., 2014,Mansouri et al., 2017]. Les autres stratégies prennent fréquemment en compte les performances comme un moyen d'activer la réplication [Zhang et al., 2015,Mokadem and Hameurlain, 2020]. Ces moyens seront discutés un peu plus tard. Cependant, la performance est représentée de différente manière, selon la popularité d'une données [Zhang et al., 2015,Liu et al., 2019], le temps de réponse

Stratégies	Objectifs		Problématiques	
	Disponibilité	Performance	Profit	Energie
[Wei et al., 2010]	✓	✗	✗	✗
[Pamies-Juarez et al., 2011]	✓	✗	✗	✗
[Selvi and Anbuselvi, 2018]	✓	✗	✗	✗
[Qu and Xiong, 2012]	✓	✓	✗	✗
[Bai et al., 2013]	✓	✓	✗	✗
[Kumar et al., 2014]	✓	✓	✗	✗
[Mansouri et al., 2017]	✓	✓	✗	✗
[Sousa et al., 2018]	✓	✓	✗	✗
[Long et al., 2014]	✓	✓	✗	✓
[Xu et al., 2015]	✓	✓	✗	✓
[Boru et al., 2015]	✗	✓	✗	✓
[Zhang et al., 2015]	✓	✓	✗	✓
[Lin and Shen, 2017]	✓	✓	✗	✓
[Alghamdi et al., 2017]	✓	✓	✗	✓
[Ebadi and Navimipour, 2019]	✓	✓	✓	✓
[Li et al., 2019]	✓	✓	✓	✓
[Li et al., 2011]	✓	✗	✓	✗
[Janpet and Wen, 2013]	✓	✓	✓	✗
[Gill and Singh, 2016]	✓	✓	✓	✗
[Tos et al., 2017]	✓	✓	✓	✗
[Liu et al., 2019]	✓	✓	✓	✗
[Mansouri and Buyya, 2019]	✓	✓	✓	✗
[Hsu and Kshemkalyani, 2019]	✓	✓	✓	✗
[Mokadem and Hameurlain, 2020]	✓	✓	✓	✗

TABLE 2.1 – Principaux objectifs visés des stratégies proposées

global [Tos et al., 2017] ou le temps de transfert d'une donnée [Mansouri and Buyya, 2019], qui implique ou non des objectifs de niveau de service. Ainsi, chaque stratégie a sa manière de considérer et d'estimer ces indicateurs de performance.

2.3.3 Classification par problématique

En plus des objectifs liés à la réplication de données, certaines stratégies tiennent compte aussi d'autres problématiques. En effet, certaines de ces stratégies considèrent les problématiques de consommation énergétique et de dépenses.

2.3.3.1 Réduire la consommation énergétique

Dans la suite de cette partie, nous étudions les stratégies qui se rapprochent plus de nos objectifs en considérant la consommation énergétique ou la dépense. Ici, nous nous concentrons sur les stratégies qui prennent en compte uniquement la consommation énergétique.

Dans la stratégie statique proposée par [Long et al., 2014] plusieurs objectifs sont pris en compte, dont la consommation énergétique. Le premier objectif de cette stratégie est la disponibilité qui se base sur la probabilité de panne d'un nœud. Puis, elle considère des problématiques de temps de réponse à partir de la taille des données, de fréquence d'accès et de capacité de transfert du nœud. L'équilibre des charges est aussi pris en compte dans les objectifs en calculant la variance des charges sur l'ensemble des nœuds. Le quatrième objectif correspond à la consommation énergétique qui est modélisée avec d'un coté la consommation des nœuds lors de l'exécution de tâches et de l'autre la consommation énergétique liée au refroidissement des nœuds. Enfin, la stratégie considère

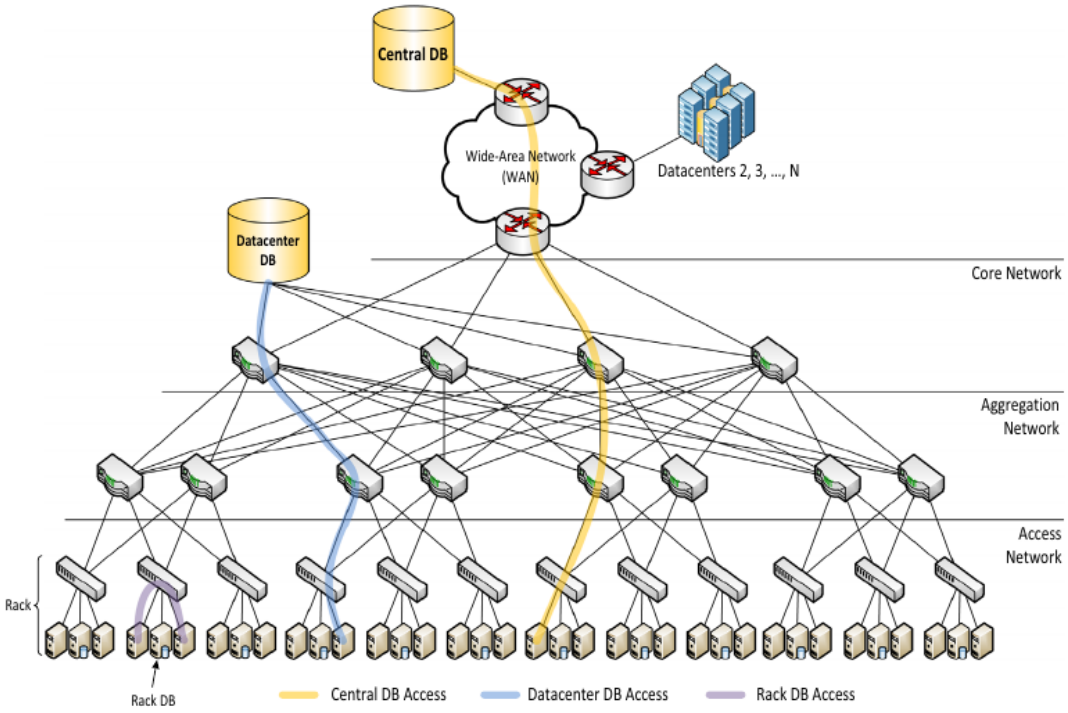


FIGURE 2.2 – Architecture adoptée dans la stratégie [Boru et al., 2015]

un objectif de latence afin de placer les données au mieux pour la minimiser et donc maximiser la bande-passante. Pour répondre à cet ensemble d'objectifs, les auteurs proposent un algorithme évolutionnaire qui applique un poids à chaque objectif.

Une autre stratégie qui met plus en avant les problématiques énergétiques est [Boru et al., 2015]. Les auteurs de cette stratégie dynamique mettent en avant une modélisation de la consommation énergétique et de la bande-passante dans le cadre d'une architecture hiérarchique. L'architecture utilisée dans cette stratégie est illustrée dans la figure 2.2. Dans cette stratégie, si le nombre d'accès moyen dans un intervalle de temps dépasse un seuil, la donnée est répliquée lorsque les bases de données des niveaux plus bas (centre de données, rack) consomment moins de bande-passante et d'énergie que les bases de données de niveaux supérieurs (centrale, centre de données).

Dans la stratégie proposée par [Zhang et al., 2015], les auteurs prennent en compte la consommation énergétique tout en tenant compte des performances. Pour cela, les auteurs mettent en place deux listes LRU (*Least Recently Used*) qui permettent de classifier les données selon leurs chaleurs, lorsqu'une donnée très chaude (en tête de liste chaude) est requêtée alors elle est répliquée. L'objectif étant de mettre dans un état moins énergivore les nœuds qui ne contiennent pas de données chaudes, lorsqu'un nœud contient une donnée chaude qui sera répliquée, elle le sera sur un nœud chaud. Les noeuds ayant des données froides sont donc mis dans un état qui consomme moins énergie. Quand une donnée devient chaude sur un nœud froid, la donnée sera répliquée sur un nœud chaud. A l'inverse, lorsqu'une donnée devient froide, elle est placée dans un nœud qui réduit sa consommation. Cependant, les nœuds froids qui contiennent un grand nombre de données chaudes deviennent chauds en devenant plus énergivores et vice versa.

Comme pour la stratégie précédente, les auteur et autrice de [Lin and Shen, 2017] proposent une stratégie qui classe les données selon qu'elles soient chaudes ou froides. Pour cela, la stratégie regarde le nombre d'accès à une réplique, ou à la donnée dans sa globalité (en considérant ses répliques). Si ces nombres dépassent un certain seuil, la donnée est considérée comme chaude. Ainsi, lorsque les données sont chaudes, sur des nœuds surchargés, alors la stratégie va créer une

nouvelle réplique qui sera placée de manière aléatoire. Elle augmente la probabilité des nœuds selon la consommation de la bande-passante et de la capacité du nœud. De la même manière que la stratégie précédente, l'objectif est d'utiliser les états inactifs des nœuds pour réduire la consommation énergétique. L'auteur et autrice remarquent que si un nœud froid doit récupérer des données, le nœud doit s'allumer, récupérer les données et s'éteindre. L'idée ici est donc de définir un troisième type de nœuds qui correspond au fait de devenir froid. Lorsqu'une donnée devient froide, soit elle est supprimée d'un nœud chaud car il y a suffisamment de répliques, soit la donnée est migrée sur un nœud qui va s'éteindre. Le nœud restera ainsi en activité jusqu'à ce que le nœud soit rempli de données froides pour qu'il s'éteigne.

Dans [Alghamdi et al., 2017], les auteurs proposent une stratégie de réplication de données, qui cherche à maximiser le profit. Il est calculé en faisant la différence de consommation énergétique générée par l'exécution de toutes les tâches sans réplication de donnée, et la consommation de cette même exécution avec réplication. A l'aide d'un algorithme glouton, la stratégie estime le profit de chaque ajout d'une réplique sur chaque nœud, si ce dernier est profitable par rapport à l'état précédent, alors cette donnée est stockée sur le nœud qui réduit la consommation énergétique. Il est à noter ici que le profit est uniquement considéré comme une amélioration du point de vue énergétique et non d'un point de vue économique.

Une autre manière de considérer la consommation énergétique est proposée dans [Xu et al., 2015]. L'objectif de cette stratégie est de minimiser l'émission de gaz à effet de serre en s'appuyant sur la réduction de la consommation énergétique et de la géo-distribution des données pour réduire cette émission à l'aide des différents mix énergétique. Lorsqu'une nouvelle donnée arrive, une réplication par défaut à l'aide d'une méthode de hachage, nommée *consistent hashing*, est mise en place sur les sites à faible émission en se basant sur une estimation des futures charges de travail. La stratégie estime le nombre optimal de répliques pour minimiser l'émission de gaz à effet de serre. Le placement de ces répliques se fait en utilisant différents algorithmes d'optimisation qui remplissent petit à petit les nœuds tant que cela minimise l'émission de CO2. Enfin, sachant que le nombre d'accès diminue, la stratégie met en place un système d'archivage.

2.3.3.2 Réduire la dépense

Dans cette partie, nous nous concentrons sur les stratégies qui considèrent les problématiques économiques sans tenir compte de la consommation énergétique.

Une stratégie proposée par [Li et al., 2011] s'intéresse au coût du stockage en se basant sur la probabilité de panne d'un nœud. L'idée principale est que la panne d'un nœud suit une loi exponentielle sur le temps d'utilisation. Ce qui implique que la probabilité de panne augmente dans le temps. Ainsi, en considérant une exigence de fiabilité, les auteurs estiment la probabilité de non-accessibilité d'une donnée selon son temps de stockage, lorsque cette probabilité est en-dessous de l'exigence pour un temps t, alors une nouvelle réplique est créée. La stratégie réduit donc les coûts de stockage en retardant au maximum la création de nouvelles répliques.

La stratégie proposée par [Janpet and Wen, 2013] cherche principalement à minimiser le temps de réponse de chaque requête. La stratégie s'intéresse ainsi au chemin pris par les données pour répondre aux requêtes afin de réduire le temps de réponse. Ainsi, si une réduction est possible en dupliquant la donnée sur le centre de données qui reçoit le plus de requêtes, alors la stratégie estime le coût de la duplication qui doit être inférieur au budget alloué à l'utilisateur ou utilisatrice. Il est important de noter que malgré ce budget un nombre minimum de répliques sont créées afin de répondre à un besoin minimum de disponibilité.

Les auteurs de [Tos et al., 2017] proposent une stratégie dynamique de réplication de données qui cherche à minimiser le nombre de violations d'objectif de niveau de service, tout en tenant compte du profit du fournisseur. Lorsque le temps de réponse est supérieur au niveau de service défini dans le SLA, la décision de répliquer se met en place. Pour cela, la stratégie estime le profit du fournisseur sur la base des dépenses et des revenus du fournisseur. Lorsque le fournisseur est profitable, alors la donnée ayant généré la violation est répliquée sur le nœud qui a exécuté la requête. De plus, un

mécanisme de suppression de répliques est mis en place lorsqu'un certain nombre de requêtes ont été exécutées sans violation.

Une autre stratégie qui prend en compte les problématiques de performance au travers des objectifs de niveaux de services est proposée par [Mokadem and Hameurlain, 2020]. Un objectif de disponibilité est à atteindre en se basant sur un nombre minimum de répliques avant l'application de la stratégie. Ensuite, la stratégie répond à des objectifs de performances qui correspondent au temps de réponse pour chaque requête. Ainsi, si ces objectifs ne sont pas atteints, la stratégie détermine la raison de la violation et réplique si elle génère du profit. Les violations peuvent être liées à la surcharge d'un nœud, ou à un goulot d'étranglement au niveau du réseau. Dans le premier cas, le but est de répliquer la donnée sur un nœud moins chargé pour mieux répartir la charge de travail. Dans le second cas, la stratégie va répliquer sur un nœud plus proche, en terme de bande passante, du nœud qui génère les violations. Un mécanisme de suppression de répliques est là aussi mis en place, qui dans un premier temps va compresser les données, puis les supprimer si elles ne sont pas requêtées.

Une stratégie qui s'intéresse à l'hétérogénéité entre les centres de données est proposée par [Gill and Singh, 2016]. Cette stratégie considère une architecture hiérarchique circulaire avec au centre un centre de données de haute qualité qui est performant, avec beaucoup de stockage, mais cher à l'utilisation ; au bord des centres de données ordinaires, moins fiables, moins performants et donc moins chers. La stratégie repose sur des calculs de score de disponibilité et de réplicabilité, qui se basent sur la probabilité de panne d'un nœud et sur le nombre d'accès récents respectivement. Le score de disponibilité doit être supérieur à un certain seuil, sinon la stratégie va répliquer et placer les données pour l'atteindre. Le score de réplicabilité permet d'estimer le nombre de répliques nécessaires pour répondre aux accès récents. Ainsi, le nombre de répliques à créer correspond à la différence entre le score de l'intervalle de temps précédent, et le score actuel. Les nouvelles répliques créées sont placées dans les centres de données de niveaux inférieurs selon le score de réplicabilité par centre de données. C'est dans ce cadre que l'aspect économique apparaît en considérant un budget de réplication, si ce budget est dépassé, alors un algorithme d'optimisation est mis en place pour réduire les coûts.

[Liu et al., 2019] est une autre stratégie qui considère les problématiques de performances d'une autre manière. Pour cela, la stratégie classe les données de 2 manières. La première selon la popularité des données qui permet de définir sa chaleur, et la seconde selon l'action (lecture ou écriture) la plus fréquemment faite sur la donnée. La stratégie crée 2 répliques qui sont placées sur 2 nœuds primaires et un nœud de sauvegarde. Ce placement se fait en minimisant la corrélation de panne de machine, en les plaçant sur des nœuds éloignés pour réduire l'impact d'une panne électrique dans une zone par exemple. La chaleur des données, permet de définir les supports de stockage des nœuds primaires, en favorisant les SSD pour les nœuds très populaires. Le classement sur le type d'action a un impact sur le type de compression mis en place dans les nœuds de sauvegarde.

Les auteurs de [Mansouri and Buyya, 2019] considèrent leur stratégie comme un intermédiaire entre plusieurs fournisseurs de cloud, qui gèrent des centres de données de manière géo-distribuée. Cette stratégie considère durant son placement initial des problématiques de latence (définie dans le SLA comme un objectif) et des problématiques de coût qui permettent de prendre l'ensemble des décisions liées à la réplication. Lorsqu'une nouvelle donnée arrive, la stratégie calcule le nombre minimum de répliques qui permet à l'ensemble des centres de données d'accéder à cette dernière sous la contrainte de latence. Puis à chaque intervalle de temps, un algorithme de placement de répliques se met en place pour initier et mettre à jour le placement des répliques en tenant compte des coûts de stockage, de lecture, de mise à jour, de réplication et de potentielle migration. Si entre deux intervalles de temps, le placement des répliques change, un algorithme de migration se met en place. Cet algorithme permet la migration ou non d'une réplique selon que le coût d'utilisation sur l'ancien nœud soit moins élevé que le coût d'utilisation sur le nouveau nœud en considérant le coût de migration.

Une stratégie qui s'intéresse à la prédiction de la future charge de travail est proposée par [Hsu and Kshemkalyani, 2019] qui considère la problématique de la réplication au travers du fonctionnement des réseaux sociaux. Pour cela, ils utilisent le modèle *Auto-Regressive Integrated Moving*

Average (ARIMA) qui permet de prédire sur une courte période de temps le nombre de d'accès à une donnée. Pour chaque intervalle de temps, une estimation est faite pour l'intervalle suivant et envoyée à l'algorithme de réplication. Cet algorithme va donc minimiser le coût global associé à la prédiction des futures tâches en minimisant les coûts dans chaque région.

2.3.3.3 Réduire la consommation énergétique et la dépense

Dans cette dernière partie nous nous intéressons aux stratégies qui prennent en compte les dépenses et la consommation énergétique. Cependant, peu de stratégies traitent de ces problématiques simultanément.

La stratégie proposée par [Ebadi and Navimipour, 2019] est une stratégie statique qui estime la consommation énergétique et la dépense des futures lectures et écritures. Ces modèles sont regroupés en fonction objectif, et sont associés à des poids. Enfin, la stratégie s'appuie sur une heuristique hybride avec un algorithme d'optimisation par essaims particulaires dans lequel à chaque itération une recherche tabou est mise en place pour améliorer les performances.

Dans un autre contexte, les autrices de [Li et al., 2019] proposent une stratégie de réplication de données dans une architecture edge. Cette stratégie met en place, dans un premier temps, une prédiction des futures charges de travail par fichier. Cette prédiction permet d'estimer le nombre de répliques nécessaires pour répondre aux futures requêtes. La stratégie choisit les nœuds qui stockeront ces répliques à l'aide d'un algorithme de placement. Pour cela, les autrices proposent des modèles de disponibilité, de performance, de charge des nœuds et des clusters, du coût de placement et du temps de réponse afin de minimiser chacun de ces modèles. Les 2 derniers modèles correspondent aussi à des contraintes de budget et de temps de réponse. Durant cette étape, la stratégie met en place un dimensionnement des nœuds afin de réduire la consommation énergétique globale sur la base de l'utilisation des nœuds.

2.3.3.4 Analyse

Nous nous sommes concentrés ici sur les considérations énergétiques et économiques des stratégies de réplication de données. Un résumé de leurs objectifs et problématiques est présenté dans le tableau 2.1.

Dans les stratégies qui prennent en compte la consommation énergétique, il est intéressant de voir que récemment les stratégies s'intéressent à d'autres architectures où les problèmes de consommation énergétiques impliquent la prise en compte de nœuds sur batterie [Vales et al., 2019]. Nous voyons cependant que la consommation énergétique est prise en compte de manières différentes. Certaines stratégies s'appuient sur l'hétérogénéité entre les nœuds pour appuyer la réduction entre les nœuds en répliquant sur des nœuds moins énergivores [Long et al., 2014, Boru et al., 2015, Xu et al., 2015]. D'autres mettent en place des techniques de réduction de la consommation énergétique, puis réduisent la consommation énergétique en mettant des nœuds dans des états qui consomment moins d'énergie mais qui rendent plus long l'accès aux données stockées dessus [Zhang et al., 2015, Lin and Shen, 2017].

Dans les stratégies qui prennent en compte la dépense, nous voyons qu'elle est modélisée de manière incrémentale. Dans un premier temps, nous avons les stratégies qui cherchent à minimiser les dépenses de stockage et qui répliquent en tenant compte de la méthode de stockage sur les nœuds [Liu et al., 2019], ou en répliquant le plus tardivement possible [Li et al., 2011]. Les stratégies qui ajoutent la dépense associée au fait de répliquer, c'est-à-dire au coût de transfert principalement [Janpet and Wen, 2013,Gill and Singh, 2016]. Et enfin, les stratégies, qui ajoutent à cela, les dépenses associées au fonctionnement du système, en tenant compte des coûts de lecture et parfois des coûts de violations d'objectif de niveau de service [Tos et al., 2017, Mokadem and Hameurlain, 2020, Mansouri and Buyya, 2019]. Ainsi, ces stratégies se divisent entre celles qui cherchent uniquement à minimiser la dépense [Li et al., 2011, Mansouri and Buyya, 2019], et celles qui considèrent un budget qu'il ne faut pas dépasser [Gill and Singh, 2016, Tos et al., 2017, Mokadem and Hameurlain, 2020].

2.3.4 Classification par temporalité

Dans cette partie, nous allons étudier la temporalité des différentes stratégies selon leurs manières de se déclencher. En effet, dans les stratégies étudiées, le déclenchement de la stratégie et la réplication de données se fait de différentes manières. Dans un premier temps, ces stratégies se divisent en 2 grandes catégories. Les stratégies statiques d'un coté, qui déclenchent la stratégie avant l'arrivée des premières requêtes utilisateur. De l'autre coté, les stratégies dynamiques qui se déclenchent durant le fonctionnement du système.

2.3.4.1 Statique

Les stratégies de réplications statiques font les choix de quelle donnée répliquer, sur quel nœud la répliquer, à quel moment et combien de répliques à l'avance. Ces stratégies se mettent en place lors de la réception des données avant l'apparition des premières lectures et écritures.

Parmi les stratégies statiques certaines s'intéressent uniquement à un placement initial sans faire d'hypothèse sur la charge de travail future. C'est le cas de [Wei et al., 2010, Pamies-Juarez et al., 2011] qui répliquent en s'intéressant principalement aux problèmes de disponibilités. D'autres stratégies [Long et al., 2014,Ebadi and Navimipour, 2019] font des hypothèses sur les futures charges de travail et en tiennent compte pour minimiser leurs objectifs respectifs.

De la même manière, [Li et al., 2011] ne tient pas en compte de la future charge de travail, mais modélise son objectif dans le temps, et met en place sa réplication en prévoyant les futures répliques à l'avance.

Une autre possibilité par [Alghamdi et al., 2017], est de connaître à l'avance l'ensemble des tâches afin de répliquer pour prendre en compte au mieux l'ensemble des objectifs.

2.3.4.2 Dynamique

Les stratégies dynamiques correspondent à des stratégies qui prennent en compte divers indicateurs pour prédire ou réagir ; donc créer, déplacer, ou supprimer des répliques. Dans le cadre de ces stratégies, plusieurs techniques peuvent être mises en place, nous étudierons ici : i) les seuils, ii) les scores, iii) les listes, iv) les intervalles de temps et enfin v) la prédiction.

i) seuils : Dans un premier temps, nous nous concentrons sur des stratégies qui s'appuient sur des seuils. Les indicateurs sur lesquels se basent les seuils diffèrent d'une stratégie à l'autre. Les stratégies proposées dans [Qu and Xiong, 2012, Boru et al., 2015, Lin and Shen, 2017, Selvi and Anbuselvi, 2018] considèrent le nombre moyen d'accès sur un intervalle de temps. Si cette moyenne dépasse un certain seuil, alors la stratégie s'applique. Dans la stratégie proposée dans [Bai et al., 2013], elle estime le temps de réponse maximum et moyen de chaque donnée, si ce temps moyen dépasse un certain seuil, alors le reste de la stratégie s'applique. Un autre moyen de prendre en compte le temps de réponse est proposée par [Tos et al., 2017] où avant chaque lecture, la stratégie estime le temps de réponse de la requête, s'il est supérieur à un seuil alors la réplication est déclenchée. Une proposition qui étend cette idée est introduite par [Mokadem and Hameurlain, 2020]. Cette stratégie s'appuie dans un premier temps sur le seuil critique présenté précédemment. Et ajoute un seuil avec une valeur moins élevée, qui déclenche la réplication si le temps de réponse dépasse ce seuil un nombre consécutif de fois. Ces stratégies s'appuient sur le temps de réponse comme seuil, permettent de mettre en place des objectifs de niveau de service sur le temps de réponse. Enfin, un dernier seuil proposé par [Kumar et al., 2014] est un seuil de déséquilibre. La stratégie calcule un score de déséquilibre qui ne doit pas dépasser la capacité du cluster sur lequel sont stockées les partitions. Une des problématiques liée au seuil est sa binarité qui fait que soit le problème n'est pas là (donc le seuil désactivé), soit il est déjà là (le seuil est activé). Certaines stratégies pallient à ce problème en multipliant les seuils comme [Mokadem and Hameurlain, 2020].

ii) scores : D'autres stratégies se basent uniquement sur un score calculé à chaque intervalle de temps pour répliquer. Les auteurs de [Gill and Singh, 2016] calculent un score de réplicabilité, sur la base des accès récents, à chaque intervalle de temps, et adaptent le nombre de répliques à ce facteur. Ainsi, si le nombre de répliques est inférieur à ce facteur, alors la stratégie va créer de nouvelle répliques. Á l'inverse si le nombre de répliques est supérieur à ce facteur alors la stratégie va en supprimer. Une autre manière de considérer un score est proposée par [Mansouri et al., 2017], qui calcule un score sur le nombre d'accès de chaque donnée, et met en place une liste dont la taille dépend du nombre de données dont l'accès a été demandé par un autre nœud que celui où la donnée est stockée. Chaque donnée est classée selon son score de nombre d'accès et celles qui sont les plus accédées sont à répliquer.

iii) listes : La stratégie proposée par [Zhang et al., 2015] met en place des listes permettant d'activer la réplication. Comme cela a été présentée plus haut, la stratégie se base sur 2 listes LRU (*Least Recently Used*). La première est la liste récente, qui correspond aux données récemment accédées. La seconde est la liste chaude, qui correspond aux données chaudes. Ainsi, lorsqu'une donnée est requêtée elle apparaît dans la liste récente, si elle est requêtée à nouveau elle se déplace à la tête de la liste chaude. Si d'autres données sont requêtées alors la donnée recule dans la liste récente/chaude. Les données chaudes requêtées qui ne sont plus en tête de liste chaude, reviennent à sa tête. Enfin, si la donnée requêtée est déjà en tête de liste chaude, alors la réplication s'active.

iv) intervalles de temps : Une autre catégorie de stratégies s'applique à chaque intervalle de temps, en mettant en place des algorithmes d'optimisation pour tenir compte de leurs objectifs. Dans la stratégie proposée par [Janpet and Wen, 2013], à chaque intervalle de temps, la stratégie s'applique afin de rapprocher les données des nœuds où elles sont requêtées sous la contrainte du budget. De même, les auteurs de [Mansouri and Buyya, 2019] appliquent leur stratégie à chaque intervalle de temps pour placer au mieux les données selon les informations récupérées durant cet intervalle de temps. Ainsi, si l'algorithme réplique et place les données différemment par rapport à l'état actuel du système, alors un algorithme de migration se met en place pour tenir compte des objectifs de la stratégie.

v) prédictions Enfin, une dernière catégorie de déclencheur est étudiée ici, ce sont les stratégies qui se basent sur des prédictions pour répliquer en tenant compte de leurs objectifs. Parmi ces stratégies, les auteurs de [Hsu and Kshemkalyani, 2019] se basent sur le modèle ARIMA (*Auto-Regressive Integrated Moving Average*) pour estimer les futures charges de travail et ainsi mettre en place la réplication pour s'adapter à cette future charge. D'autres stratégies estiment le nombre d'accès des intervalles suivants pour chaque donnée. Ainsi, la stratégie proposée dans [Li et al., 2019], s'appuie sur cette prédiction pour calculer un score de réplication pour l'intervalle suivant. Dans une idée un peu similaire, [Xu et al., 2015] essaie d'approcher les lois de probabilité des accès de chaque objet. Pour cela, il estime les paramètres de ces lois en s'appuyant sur un petit nombre de requêtes. La qualité d'approximation se mesure en utilisant un test de Kolmogorov-Smirnov. Une autre prédiction est faite sur le mix énergétique des centres de données afin de prédire les futures émissions de gaz à effet de serre. En se basant sur ces 2 prédictions, la stratégie va répliquer de sorte à prendre en compte ses objectifs pour le prochain intervalle de temps. Dans la stratégie proposée par [Liu et al., 2019], elle estime la popularité des données pour l'intervalle de temps suivant en se basant sur le nombre d'accès de chaque donnée. Ainsi, les premiers 25% des données les plus populaires sont considérées chaudes, puis les 25% suivantes sont les données tièdes, les autres données sont considérées comme froides. Un dernier type de prédiction est proposé dans [Sousa et al., 2018] qui estime le temps avant la prochaine violation de chaque donnée. Ainsi, si le temps avant la prochaine violation atteint le temps pour répliquer la donnée, alors l'algorithme va s'appliquer.

Stratégies	Statique	Dynamique				
		Intervalle de temps	Seuil	Score	Prédiction	Liste
[Wei et al., 2010]	✓	✗	✗	✗	✗	✗
[Pamies-Juarez et al., 2011]	✓	✗	✗	✗	✗	✗
[Selvi and Anbuselvi, 2018]	✗	✓	✓	✗	✗	✗
[Qu and Xiong, 2012]	✗	✓	✓	✗	✗	✗
[Bai et al., 2013]	✗	✓	✓	✗	✗	✗
[Kumar et al., 2014]	✗	✓	✓	✗	✗	✗
[Mansouri et al., 2017]	✗	✓	✗	✓	✗	✗
[Sousa et al., 2018]	✗	✓	✗	✗	✓	✗
[Long et al., 2014]	✓	✗	✗	✗	✗	✗
[Xu et al., 2015]	✗	✓	✗	✗	✓	✗
[Boru et al., 2015]	✗	✓	✓	✗	✗	✗
[Zhang et al., 2015]	✗	✗	✗	✗	✗	✓
[Lin and Shen, 2017]	✗	✓	✓	✗	✗	✗
[Alghamdi et al., 2017]	✓	✗	✗	✗	✗	✗
[Ebadi and Navimipour, 2019]	✓	✗	✗	✗	✗	✗
[Li et al., 2019]	✗	✓	✗	✓	✓	✗
[Li et al., 2011]	✓	✗	✗	✗	✗	✗
[Janpet and Wen, 2013]	✗	✓	✗	✗	✗	✗
[Gill and Singh, 2016]	✗	✓	✗	✓	✗	✗
[Tos et al., 2017]	✗	✗	✓	✗	✗	✗
[Liu et al., 2019]	✗	✓	✗	✓	✓	✓
[Mansouri and Buyya, 2019]	✗	✓	✗	✗	✗	✗
[Hsu and Kshemkalyani, 2019]	✗	✓	✗	✗	✓	✗
[Mokadem and Hameurlain, 2020]	✗	✗	✓	✗	✗	✗

TABLE 2.2 – Techniques d'activation des stratégies de réplication de données

2.3.4.3 Analyse

Un résumé des techniques étudiées dans cet état de l'art est présenté dans le tableau 2.2.

Dans un premier temps, nous avons vu les stratégies qui sont statiques, il est intéressant de noter que certaines stratégies dynamiques mettent en place un placement initial qui dépendent de différents objectifs.

De plus, nous voyons que beaucoup de stratégies s'appuient sur des intervalles de temps pour mettre en place les déclencheurs [Janpet and Wen, 2013, Xu et al., 2015]. Certaines stratégies calculent leurs indicateurs à l'aide des connaissances acquises durant l'intervalle de temps, ce qui permet d'activer ou non la réplication [Mansouri et al., 2017, Li et al., 2019]. Cependant, si cet intervalle est trop long, alors le temps d'adaptation à de forte augmentation ou baisse de charge se fera en retard. Et à l'inverse, si ce temps est trop court, alors il n'y aura pas assez de données pour calculer ces indicateurs, et le coût d'exécution de la stratégie deviendra trop élevé.

Peu de stratégies dynamiques s'appuient sur autre chose que des intervalles de temps, comme [Tos et al., 2017, Mokadem and Hameurlain, 2020] qui s'appuient sur un déclenchement par lecture, ou par groupe de lectures. Comme dans le cadre des intervalles, estimer le temps de réponse afin de déclencher la stratégie à chaque lecture a un impact élevé en terme de coût de calcul. Mais cela a comme avantage de pouvoir s'adapter rapidement à une augmentation de charge de travail.

2.4 Conclusion

Nous avons vu dans un premier temps dans cet état de l'art, que peu de stratégies s'intéressent à la fois à la consommation énergétique et à la dépense. De plus, dans les stratégies qui considèrent la consommation énergétique, seulement une petite partie de ces stratégies s'appuient sur des techniques de réduction de consommation énergétique. De même, une partie des stratégies qui s'intéressent aux dépenses comparent cela à un budget ou revenu, qui bloque la réplication si cette dépense est supérieure au revenu pouvant ainsi empirer les dépenses.

Dans ce cadre, la stratégie proposée dans cette thèse a pour principales problématiques la consommation énergétique et la dépense du fournisseur. Pour tenir compte de ces problématiques, nous nous appuyons à la fois sur l'hétérogénéité entre les nœuds mais aussi en utilisant des techniques de réduction de consommation énergétique. De plus, la stratégie considérera un placement initial et une réplication dynamique dont l'objectif sera de créer et supprimer des répliques.

Dans les déclencheurs des stratégies de réplication, une minorité de stratégies s'appuient sur d'autres techniques que des intervalles de temps pour déclencher la stratégie. Parmi ces déclencheurs, une partie de ces stratégies impliquent une estimation du temps de réponse avant chaque lecture ce qui est coûteux.

L'idée de notre stratégie est de tenir compte des violations de niveau de service pour s'activer, tout en mettant en place un déclencheur ayant un faible coût de calcul. Ces violations permettront de récolter des informations pour la stratégie dynamique.

Dans le chapitre suivant, nous allons introduire les concepts nécessaires à la bonne compréhension de la stratégie de réplication de données proposée.

Chapitre 3

Contexte

Sommaire

3.1 Introduction

Ce chapitre de contexte permet d'introduire les outils et modèles utilisés dans la stratégie de réplication de données proposée.

Elle se compose d'un placement initial et d'une gestion dynamique de répliques. La partie 3.2 présentera les modèles de consommation énergétique et de dépenses considérés, provenant de la littérature. Ces modèles sont principalement utilisés dans le placement initial pour être articulés en *fonctions objectifs* nécessaires aux algorithmes d'optimisation.

Dans la partie 3.3, nous introduirons les outils mathématiques utilisés dans les chapitres suivants. Nous allons ainsi présenter les informations fondamentales liées aux algorithmes d'optimisation multi-objectif utilisées dans le placement initial. Ce sera suivi d'une présentation des cartes de contrôles qui permettront d'activer la réplication.

3.2 Modélisation

Nous introduisons dans cette partie, les modèles de consommation énergétique et de dépense. Nous nous servirons de ces modèles dans la stratégie de réplication proposée, mais aussi comme métriques pour comparer la stratégie proposée avec d'autres stratégies issues de la littérature.

Dans un premier temps, des notations seront introduites afin de faciliter la compréhension des modèles présentés. Puis la modélisation de la consommation énergétique se décomposera en modèle par composant, puis en modèles par action qui met en avant les interactions des composants. Enfin, nous terminerons par les modèles de dépenses utilisés dans la stratégie et dans les évaluations.

3.2.1 Notation

Avant de présenter les modèles de consommation énergétique et de dépenses, nous allons introduire quelques notations. Nous allons dans un premier temps (i) décrire les nœuds avec leurs composants qui permettent d'introduire les notations associées à ces composants. Puis, (ii) les commutateurs réseaux qui permettent de décrire les équipements utilisés pour transférer les données. Ensuite, seront introduites les notations associées (iii) aux données et (iv) aux tâches. Nous terminerons par les notations liées aux (v) actions nécessaires pour l'exécution de tâches et la réplication de données. Un résumé de l'ensemble des notations introduites dans ce chapitre est présenté dans le tableau 3.1 à la fin du chapitre.

3.2.1.1 Les nœuds

Soit N représentant un ensemble de nœuds. Chaque nœud, noté $n_j \in N$, $1 \leq j \leq m$ contient plusieurs caractéristiques matérielles. En effet, nous supposons que chaque nœud peut à la fois stocker des données et exécuter des tâches. Pour cela, il est composé (i) d'un **processeur**, (ii) d'une **mémoire**, (iii) d'un **disque** et (iv) d'une **carte réseau**.

Le processeur a un pourcentage d'utilisation u_j et fonctionne à une fréquence fr_j (en hertz) qui impacte la puissance maximale, notée $P^{fr_j}_{CPU_{max}}$, et en inactivité, notée $P^{fr_j}_{CPU_{idle}}$.

La mémoire a une capacité de stockage cp^j_{RAM} (Go), une bande-passante de transfert de données B^j_{RAM} (Go/s), et une puissance selon son état $P^j_{RAM_{state}}$. La mémoire peut avoir les états de lecture, d'écriture et d'inactivité : $state \in \{idle, read, write\}$

Le stockage sur disque a une capacité cp^j_{disk} (Go), un temps de recherche moyen t^j_{seek} (en seconde), une bande-passante B^j_{disk} (Go/s) et une consommation en fonction de son état $P^j_{disk_{state}}$. Le stockage peut avoir les états de recherche de données, d'activité, et d'inactivité : $state \in \{seek, active, idle\}$.

La carte réseau a une bande-passante B^j_{NIC} (Go/s), et comme le processeur, une consommation active et inactive notées $P^j_{NIC_{active}}$ et $P^j_{NIC_{idle}}$ respectivement.

Enfin, une dernière notation est T^j_{tot} (s) correspondant au temps total d'utilisation du nœud j.

3.2.1.2 Commutateurs réseaux

Soit SW représentant un ensemble de commutateurs réseaux qui permettent de connecter les nœuds entre eux. Chaque commutateur $sw_p \in SW$, $1 \leq p \leq q$ a une bande-passante B^p_{SW} (Go/s) et possède un nombre de ports $Port^p$. Les commutateurs sont placés à différents niveaux. Un niveau de commutateur région ($Region$), qui connecte les régions entre elles, et les centres de données d'une même région. Et un second niveau de commutateur centre de données (DC), qui connecte les nœuds d'un même centre de données, et se connecte vers le commutateur de sa région. Le niveau est noté : $lvl(sw_p) \in \{DC, Region\}$. Chaque commutateur connecte un ensemble de nœuds notés : $nodes(sw_p) \subset N$. Enfin, les puissances du commutateur en activité et inactivité sont notées $P^p_{SW_{active}}$ et $P^p_{SW_{idle}}$ respectivement.

3.2.1.3 Les données

Soit F représentant un ensemble de données stockées sur un ensemble de nœuds N. Chaque donnée $f_i \in F$, $1 \leq i \leq z$ a une taille $s(f_i)$ (Go).

Soit ϕ, une matrice de taille (z, m) et représentant le placement des données sur les nœuds : $\phi(f_i, n_j)$ est égale à 1 si la donnée f_i est stockée sur le nœud n_j, et 0 sinon.

3.2.1.4 Les tâches

Soit T qui représente l'ensemble des tâches qui sont à exécuter. Chaque tâche $t_k \in T$, $1 \leq k \leq y$ a un nombre d'instructions $s(t_k)$ à exécuter, et nécessite une ou plusieurs données $f(t_k) \in F$. Elle a un temps total d'exécution (en tenant compte du temps de transfert) noté $tps(t_k)$ (s) dépendant du nœud sur lequel elle est placée. Cependant, une tâche doit répondre à des objectifs de niveaux de services qui correspondent ici à un temps de réponse maximal noté tps_{SLO} (s).

3.2.1.5 Les actions

L'exécution des tâches implique diverses actions de la part des nœuds. Ces actions ont un impact **énergétique** et **économique**, et seront modélisées par la suite. Cependant, il est important d'introduire les notations associées à ces actions.

Énergie : la consommation énergétique est divisée en 2 parties. D'un coté la consommation statique, lorsque le nœud est allumé, mais inactif (*static*). De l'autre la consommation dynamique, qui correspond à la consommation énergétique pour effectuer des actions. Les actions présentes sont : (i) l'écriture d'une donnée sur un nœud, (ii) sa lecture, (iii) son transfert entre les nœuds et (iv) l'exécution d'une tâche. La consommation énergétique est ainsi notée : EC_a, $a \in \{Static, Read, Write, Net, Exec\}$.

Dépense : certaines actions ont des impacts sur la dépense du fournisseur comme (i) le stockage d'une donnée, (ii) son transfert, (iii) l'exécution d'une tâche et (iv) le coût associé à la violation de SLO. Chaque dépense est notée : EX_b, $b \in \{Storage, Network, Exec, SLO\_violation\}$. Dans ce cadre, des prix permettent de modéliser ces dépenses avec :

- $Pr^{n_j}_{Storage}$ qui correspond au prix du stockage en \$/Go par seconde d'utilisation du nœud n_j
- $Pr_{Network}(n_j, n_{j'})$ qui correspond au prix de transfert de données en \$/Go entre n_j et $n_{j'}$
- $Pr^{n_j}_{exec}$ qui correspond au prix d'exécution en \$/Instruction pour le nœud n_j

En plus de ces prix, $Revenue_{t_k}$ correspond au revenu associé à l'exécution de la tâche t_k et $Prop_{Refund}(level)$ correspond au pourcentage de pénalité sur le revenu de la tâche selon le niveau de violation. En effet, comme cela est présenté dans les contrats de niveau de service, l'accumulation de violation de SLOs entraîne des augmentations de pénalités selon différents paliers.

3.2.2 Consommation énergétique

Nous allons ici modéliser la consommation énergétique en 2 étapes. Dans un premier temps, une modélisation de la consommation énergétique par composant. Suivi d'une modélisation de la consommation énergétique de différentes actions qui se base ainsi sur les composants. L'estimation de la consommation énergétique implique une puissance pour utiliser le composant et un temps d'utilisation, elle est modélisée de la manière suivante :

$$Energy = Power * Time \tag{3.1}$$

3.2.2.1 Modélisation par composant

Dans la modélisation proposée, nous nous concentrons sur les composants qui ont le plus d'impact et sur lesquels on peut interagir.

L'article [Malladi et al., 2012], repris par [Dayarathna et al., 2016] montre que le CPU correspond à au moins 30% de la puissance de serveur, la mémoire à au moins 19%, le stockage au moins 5% et le réseaux 2%. Le reste de la puissance est consommée par le reste des composants comme la carte mère avec un pourcentage de consommation de 4% à 14% selon la configuration du serveur.

Nous allons modéliser la consommation énergétique du **processeur**, de la **mémoire**, du **disque**, de la **carte réseau** et enfin des **commutateurs réseaux**.

Il est à noter que pour la modélisation des composants, la puissance est divisée entre la puissance statique et la puissance dynamique. La puissance statique correspond à la puissance nécessaire pour maintenir un composant en inactivité. La puissance dynamique correspond à la puissance nécessaire à l'activité d'un composant lorsqu'il est utilisé.

Processeur : Le processeur est le composant qui permet l'exécution de tâches. Ces tâches sont découpées en instructions que le processeur doit exécuter, et la fréquence du processeur correspond à la vitesse à laquelle il exécute une instruction. Dans la modélisation de la consommation énergétique, le temps d'utilisation en activité correspond ainsi au temps nécessaire pour répondre à une tâche. En se basant sur [Fan et al., 2007], repris par [Liu et al., 2017], la consommation énergétique du processeur est modélisée de la manière suivante :

$$E_{CPU}(n_j, t_k) = E^{CPU}_{act}(n_j, t_k) + E^{CPU}_{idle}(n_j) \tag{3.2}$$

$$E^{CPU}_{act}(n_j, t_k) = (P^{fr_j}_{CPU_{max}} - P^{fr_j}_{CPU_{idle}}) * (2u_j - u^r_j) * \frac{s(t_k)}{fr_j} \tag{3.2a}$$

$$E^{CPU}_{idle}(n_j) = P^{fr_j}_{CPU_{idle}} * T^j_{tot} \tag{3.2b}$$

avec fr_j qui correspond à la fréquence du processeur, u_j est le pourcentage d'utilisation du processeur, r un paramètre de calibration estimé par les auteurs à 1.4 et $s(t_k)$ représente le nombre d'instructions que doit réaliser le processeur.

Mémoire : La mémoire représentée par la *Dynamic Random Access Memory* (DRAM), permet de stocker les données temporairement pour accélérer leurs accès au CPU ou à la carte réseau. Le modèle de consommation énergétique utilisé ici se base sur le modèle de puissance proposée dans [Rhu et al., 2013] mis à jour à chaque nouvelle génération de *Synchronous Dynamic Random Access Memory* (SDRAM). Ces mises à jour sont proposées par les fabricants de DRAM tel que [Micron, 2007] pour la *Double Data Rate 3* (DDR3) et [Micron, 2017] pour la DDR4.

A partir de ses feuilles de calculs et en se basant sur les Google Stress test fournis dans ces feuilles, on récupère les informations de P_{idle} ; P_{write} et P_{read}.

Sur cette base, nous proposons le modèle de consommation énergétique suivant :

$$E_{RAM}(n_j, f_i) = E^{RAM}_{read}(n_j, f_i) + E^{RAM}_{write}(n_j, f_i) + E^{RAM}_{idle}(n_j) \tag{3.3}$$

$$E^{RAM}_{read}(n_j, f_i) = (P^j_{RAM_{read}} - P^j_{RAM_{idle}}) * \frac{s(f_i)}{B^j_{RAM}} \tag{3.3a}$$

$$E^{RAM}_{write}(n_j, f_i) = (P^j_{RAM_{write}} - P^j_{RAM_{idle}}) * \frac{s(f_i)}{B^j_{RAM}} \tag{3.3b}$$

$$E^{RAM}_{idle}(n_j) = P^j_{RAM_{idle}} * T^j_{\text{tot}} \tag{3.3c}$$

Disque : Le disque dur est le composant qui permet le stockage de données sur de longues périodes. Contrairement à la mémoire RAM, le transfert de données est moins rapide et un temps de recherche de la donnée doit être pris en compte. La modélisation de la consommation énergétique de l'utilisation du disque est basée sur la proposition [Hylick and Sohan, 2009]. Ce modèle est plus récemment utilisé par [Song, 2018] pour estimer la consommation des disques *Solid-State Drive*. Le modèle utilisé est le suivant :

$$E_{disk}(n_j, f_i) = E^{disk}_{act}(n_j, f_i) + E^{disk}_{idle}(n_j) \tag{3.4}$$

$$E^{disk}_{act}(n_j, f_i) = (P^j_{disk_{Active}} - P^j_{disk_{idle}}) * \frac{s(f_i)}{B^j_{disk}} + (P^j_{disk_{Seek}} - P^j_{disk_{idle}}) * t^j_{Seek} \tag{3.4a}$$

$$E^{disk}_{idle}(n_j, f_i) = P^j_{disk_{idle}} * T^j_{\text{tot}} \tag{3.4b}$$

Carte réseau : Pour modéliser la consommation énergétique d'une carte réseau d'un nœud, nous nous sommes basés sur le document présentant le standard 802.3az [Christensen et al., 2010]. Ce standard a pour but de réduire la consommation énergétique lorsque la carte est inactive. Nous supposons ici, que les cartes réseaux sur les nœuds n'ont qu'un seul port. Le modèle utilisé est donc le suivant :

$$E_{NIC}(n_j, f_i) = E^{NIC}_{act}(n_j, f_i) + E^{NIC}_{idle}(n_j) \tag{3.5}$$

$$E^{NIC}_{act}(n_j, f_i) = (P^j_{NIC_{Active}} - P^j_{NIC_{idle}}) * \frac{s(f_i)}{B^j_{NIC}} \tag{3.5a}$$

$$E^{NIC}_{idle}(n_j) = P^j_{NIC_{idle}} * T^j_{\text{tot}} \tag{3.5b}$$

Commutateur réseaux : Dans le cadre des commutateurs réseaux, nous utilisons le même standard [Christensen et al., 2010]. Cependant, un commutateur réseaux a plusieurs ports pour connecter différents nœuds et d'autres commutateurs. Pour cela, nous nous appuyons sur [Sohan et al., 2010] qui montre que la consommation statique dépend du nombre de ports activés, augmentant la puissance totale avec l'augmentation du nombre de ports activés. Nous obtenons ainsi le modèle suivant :

$$E_{SW}(n_j, sw_p, f_i) = E^{SW}_{act}(sw_p, f_i) + E^{SW}_{idle}(n_j, sw_p) \tag{3.6}$$

$$E^{SW}_{act}(sw_p, f_i) = ((P^p_{SW_{act}} - P^p_{SW_{idle}}) * \frac{s(f_i)}{B^p_{SW}})/Port^p \tag{3.6a}$$

$$E^{SW}_{idle}(n_j, sw_p) = P^p_{SW_{idle}} * T^j_{\text{tot}} \tag{3.6b}$$

3.2.2.2 Modélisation par action

La modélisation par action a pour objectif de regrouper les modélisations par composant en fonction des actions qu'un nœud doit faire. Dans un premier temps nous regroupons les consommations statiques de tous les composants d'un nœud pour estimer (i) la **consommation statique** globale d'un nœud. Puis nous considérerons les actions (ii) d'**exécuter une tâche**, (iii) de **transférer une donnée**, (iv) de **lire une donnée**, et (v) d'**écrire une donnée**.

Consommation statique : En se basant sur la modélisation des composants, leurs parties statiques (i.e. *idle*) sont regroupées dans ce modèle. Il est à noter cependant que nous nous appuyons sur l'utilisation d'un état de veille comme le défini [Meisner et al., 2009, Wang et al., 2011]. Nous utilisons le paramètre p_{sleep} qui correspond au pourcentage de puissance restant de l'état de veille

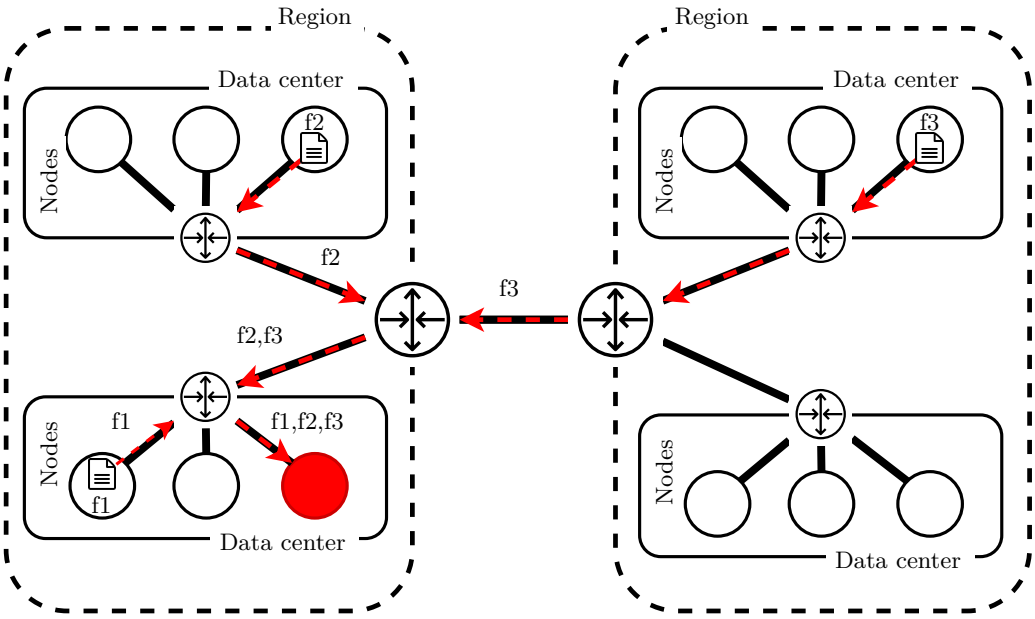


FIGURE 3.1 – Représentation des différents niveaux de transferts

par rapport à l'état inactif d'un nœud. Plus ce pourcentage est élevé, plus l'état de veille consomme. Lorsque cette valeur est nulle, la puissance nécessaire à maintenir le nœud en état de veille est la même que la puissance nécessaire pour maintenir le nœud inactif. L'état du nœud est notifié par le paramètre $State_{active}(n_j, t)$, il est égal à 1 lorsque le nœud est inactif et 0 lorsqu'il est en état de veille au temps t. Lorsqu'un nœud est en veille, il peut être allumé. Cet allumage correspond à une consommation active de l'ensemble des composants durant le temps de l'allumage $t_{sleep-active}$. Le nombre d'allumages pour chaque nœud est noté $nb^j_{activation}$. Nous obtenons ainsi le modèle suivant :

$$\begin{aligned} EC_{Static}(sw_p, n_j, T_{tot}) = E^{SW}_{idle}(n_j, sw_p) + P_{idle}(n_j) * p_{sleep} * T^j_{tot} + \\ nb^j_{activation} * t_{sleep-active} * (P_{active}(n_j) - P_{idle}(n_j)) + \\ (1 - p_{sleep}) * P_{idle}(n_j) * \int_0^{T^j_{tot}} State_{Active}(n_j, t)dt \end{aligned} \tag{3.7}$$

avec :

$$P_{idle}(n_j) = P^j_{CPU_{idle}} + P^j_{RAM_{idle}} + P^j_{disk_{idle}} + P^j_{NIC_{idle}} \tag{3.8}$$

$$P_{active}(n_j) = P^{fr_j}_{CPU_{max}} + P^j_{RAM_{read}} + P_{disk_{Active}} + P^j_{NIC_{Active}} \tag{3.9}$$

Exécution d'une tâche : Pour exécuter une tâche t_k, les données nécessaires sont lues de la mémoire RAM puis la tâche s'exécute sur le processeur. Le modèle considéré est le suivant :

$$EC_{Exec}(t_k, n_j) = E^{CPU}_{act}(n_j, t_k) + \sum_{f_i \in f(t_k)} E^{RAM}_{read}(n_j, f_i) \tag{3.10}$$

Transfert de données : Dans le cadre de la lecture et écriture d'une donnée, un transfert de donnée peut être fait entre les nœuds. Le transfert peut se faire à trois niveaux. Ces 3 niveaux sont représentés dans la figure 3.1. Le premier niveau est le transfert entre les nœuds d'un même

centre de données (noté *f1* dans la figure), où la donnée part du nœud source (n_{src}) passe par le commutateur du centre de données puis arrive sur le nœud cible (n_{tgt}). Ainsi, la consommation énergétique de ce niveau de transfert est modélisée de la manière suivante :

$$\begin{aligned} EC_{Net}(n_{src}, n_{tgt}, f_i) &= E_{act}^{SW}(sw_p, f_i) \\ avec : &\{n_{src}, n_{tgt}\} \in nodes(sw_p) \ \& \ lvl(sw_p) = DC \end{aligned} \tag{3.11}$$

Le deuxième niveau est le transfert entre les nœuds de différents centres de données d'une même région (noté *f2* dans la figure), où la donnée part du nœud source (n_{src}), passe par le commutateur réseau du centre de données de la source, puis par le commutateur réseau de la région, le commutateur du centre de données du nœud cible et termine sur le nœud cible (n_{tgt}). La consommation est modélisée :

$$\begin{aligned} EC_{Net}(n_{src}, n_{tgt}, f_i) = &E_{act}^{SW}(sw_p, f_i) + E_{act}^{SW}(sw_{p'}, f_i) + \\ &E_{act}^{SW}(sw_{p''}, f_i) \end{aligned} \tag{3.12}$$

$$\begin{aligned} avec : &\{n_{src}\} \in nodes(sw_p) \ \& \ lvl(sw_p) = DC \\ &\{n_{src}, n_{tgt}\} \in nodes(sw_{p'}) \ \& \ lvl(sw_{p'}) = Region \\ &\{n_{tgt}\} \in nodes(sw_{p''}) \ \& \ lvl(sw_{p''}) = DC \end{aligned}$$

Le dernier niveau (noté *f3* dans la figure) correspond à un transfert de données entre 2 nœuds de différentes régions. Pour cela la donnée part du nœud source (n_{src}), passe par le commutateur réseaux de son centre de données puis de sa région, descend par le commutateur de la région ensuite du centre de donnée cible pour finir dans le nœud cible (n_{tgt}). Le modèle de consommation énergétique de ce transfert est le suivant :

$$\begin{aligned} EC_{Net}(n_{src}, n_{tgt}, f_i) = &E_{act}^{SW}(sw_p, f_i) + E_{act}^{SW}(sw_{p'}, f_i) + \\ &E_{act}^{SW}(sw_{p''}, f_i) + E_{act}^{SW}(sw_{p'''}, f_i) \end{aligned} \tag{3.13}$$

$$\begin{aligned} avec : &\{n_{src}\} \in nodes(sw_p) \ \& \ lvl(sw_p) = DC \\ &\{n_{src}\} \in nodes(sw_{p'}) \ \& \ lvl(sw_{p'}) = Region \\ &\{n_{tgt}\} \in nodes(sw_{p''}) \ \& \ lvl(sw_{p''}) = Region \\ &\{n_{tgt}\} \in nodes(sw_{p'''}) \ \& \ lvl(sw_{p'''}) = DC \end{aligned}$$

Lire une donnée : Les données nécessaires à l'exécution d'une tâche peuvent être placées sur le nœud qui exécute la tâche ($src = tgt$). Dans ce cas, il est seulement nécessaire de lire la donnée sur le disque et de l'écrire en mémoire. Sinon, la donnée est placée sur un autre nœud et il faut tenir compte de la consommation énergétique associée au transfert de la donnée entre la source (src) et le nœud qui exécute la tâche (tgt). Cependant, dans le cadre de la réplication de données, plusieurs nœuds peuvent stocker une donnée. Ainsi, pour une donnée f_i, le nœud de stockage qui sera sélectionné est le plus proche en terme de bande-passante noté $short(n_{src}, n_{tgt}, f_i)$ qui est égal à 1 si n_{src} est le nœud le plus proche de n_{tgt}.

$$EC_{Read}(N, n_{tgt}, f_i) = \sum_{n_{src} \in N} \phi(n_{src}, f_i) * short(n_{src}, n_{tgt}, f_i) * [\\ E^{disk}_{read}(n_{src}, f_i) + \mathbb{1}_{src \neq tgt}(E^{RAM}_{write}(n_{src}, f_i) + E^{RAM}_{read}(n_{src}, f_i) + \\ E^{NIC}_{active}(n_{src}, f_i) + EC_{Net}(n_{src}, n_{tgt}, f_i) + E^{NIC}_{active}(n_{tgt}, f_i)) + E^{RAM}_{write}(n_{tgt}, f_i)] \tag{3.14}$$

Écrire une donnée : Lors de la réplication d'une donnée par exemple, la donnée est lue d'une mémoire RAM pour être écrite sur le disque. Cependant, comme pour la lecture d'une donnée, il est possible que le nœud possédant la donnée en mémoire soit différent du nœud cible sur lequel doit être écrite la donnée. Il faut donc tenir compte du transfert de la donnée, ce qui implique la réutilisation de la fonction $short(n_{src}, n_{tgt}, f_i)$ présentée dans la partie précédente.

$$EC_{write}(n_{src}, n_{tgt}, f_i) = E^{RAM}_{read}(n_{src}, f_i) + \\ \mathbb{1}_{src \neq tgt}[E^{NIC}_{act}(n_{src}, f_i) + E_{Net}(n_{src}, n_{tgt}, f_i) + E^{NIC}_{active}(n_{tgt}, f_i) + \\ E^{RAM}_{write}(n_{tgt}, f_i) + E^{RAM}_{read}(n_{tgt}, f_i)] + E^{disk}_{write}(n_{tgt}, f_i) \tag{3.15}$$

3.2.3 Dépense

Enfin, nous terminons la partie modélisation par les dépenses du fournisseur. Ces modèles sont basés sur ceux proposés dans [Tos et al., 2017]. Nous reprenons les modèles de dépense liée au (i) stockage et (ii) transfert de données, (iii) l'exécution d'une tâche, et (iv) la dépense liée aux violations de SLO.

3.2.3.1 Stockage de données

La modélisation de la dépense liée au stockage, se base sur la taille du fichier à stocker, du temps de stockage de la donnée sur le nœud t_{stor}, et surtout du prix de stockage qui dépend de la localisation du nœud n_j, noté $Pr^{n_j}_{Storage}$:

$$EX_{\text{Storage}}(n_j, f_i, t_{stor}) = Pr^{n_j}_{Storage} * s(f_i) * t_{stor} \tag{3.16}$$

3.2.3.2 Transfert de données

Pour le transfert d'une donnée, la dépense dépend surtout du niveau de transfert. En effet, les transferts dans un même centre de données coûtent moins chers que des transferts entre centres de données d'une même région qui restent moins chers qu'un transfert entre régions. Les coûts de transfert sont modélisés de la manière suivante :

$$EX_{\text{Network}}(n_{src}, n_{tgt}, f_i) = Pr_{Network}(n_{src}, n_{tgt}) * s(f_i) \tag{3.17}$$

3.2.3.3 Exécution de tâches

Dans les architectures Cloud, les tâches exécutées par les utilisateurs et utilisatrices ont un coût modélisé de la manière suivante :

$$EX_{\text{Exec}}(n_j, t_k) = Pr^{n_j}_{Exec} * s(t_k) \tag{3.18}$$

3.2.3.4 Violation de SLA

Enfin, une dernière dépense modélisée est la pénalité associée au fait que le temps d'exécution d'une tâche dépasse l'objectif de niveau de services défini dans le *SLA*. La pénalité associée à ces violations correspond à des niveaux de violations décrit dans les SLA. En effet, si le pourcentage de violation, parmi le nombre de tâches exécutées, dépasse des seuils pour un même usager, alors la pénalité augmente. Par exemple, dans le Cloud de Google chaque violation correspond à un remboursement de 10% du coût de locations, et lorsque le pourcentage de violation dépasse le seuil de 1% alors ce coût monte à 25% pour atteindre 100% lorsque le pourcentage de violation dépasse le seuil de 5%. Les niveaux de remboursement (*level* dans la formule) correspondent à ces différents niveaux liés aux violations.

$$EX_{\text{SLO\_violation}}(n_j, t_k) = \mathbb{1}_{tps(t_k) > tps_{SLO}} * Revenue_{t_k} * Prop_{Refund}(level) \tag{3.19}$$
avec : level $\in \{1, 2, 3, \dots\}$ selon le nombre de niveaux de remboursements

3.3 Outils Mathématiques

Nous allons présenter les outils mathématiques utilisés dans les chapitres suivants. Dans un premier temps, nous présentons les concepts associés à l'optimisation multi-objectif. Puis, nous terminerons par une description du fonctionnement des cartes de contrôle.

3.3.1 Optimisation multi-objectif

La stratégie de réplication que nous proposons dans ce manuscrit cherche à réduire la consommation énergétique dans un contexte Cloud. Nous considérons notre stratégie comme orientée fournisseur, c'est-à-dire que nous supposons être le fournisseur d'une plateforme de Cloud, impliquant des problématiques économiques. Or, ces objectifs ne sont pas forcément liés et il est possible de réduire la consommation énergétique sans pour autant réduire la dépense.

Comme nous l'avons introduit précédemment, la stratégie de réplication considère un placement initial des répliques suivi d'une gestion dynamique des répliques. C'est dans le placement initial que nous utilisons les algorithmes d'optimisation multi-objectif.

Une introduction à l'optimisation multi-objectif est proposée par [Deb, 2014] reprise par [Grange, 2019]. Cette introduction à l'optimisation multi-objectif, n'a pas pour but d'être exhaustive, mais de permettre de fournir quelques concepts nécessaires à la bonne compréhension du reste du manuscrit.

Les problèmes auxquels répond cette catégorie d'optimisation, tiennent compte de M objectifs et doivent respecter J contraintes. Chaque objectif est formalisé avec une fonction objectif, qui permet d'évaluer une proposition de solution au regard de ce dernier. De même, les contraintes correspondent à des équations ou inéquations que doivent remplir les propositions, pour qu'elles soient considérées comme solutions. Le problème est écrit de la manière suivante :

$$Minimise\ f_m(x) \quad \forall m \in 1, \dots, M \tag{3.20}$$
$$Sous\ Contrainte\ g_j(x) \geq 0 \quad \forall j \in 1, \dots, J \tag{3.21}$$

Avec $x \in \mathbb{R}^n$ qui est une solution, s'il satisfait toutes les contraintes g_j. L'ensemble de ces fonctions objectifs forment l'espace d'objectif, noté $Z \subseteq \mathbb{R}^M$. Cet espace permet de représenter chaque solution selon son impact sur chaque objectif. Pour cela, chaque solution est associée à un vecteur objectif $f(x) = (f_1(x), \dots, f_M(x))$, qui représente cette solution dans Z. L'ensemble des solutions forment un espace de solution, noté $S \subseteq \mathbb{R}^n$.

Toute la problématique de l'optimisation multi-objectif est donc de choisir une solution de cet espace. Une manière d'extraire de cet ensemble de solutions, un sous-ensemble plus adapté

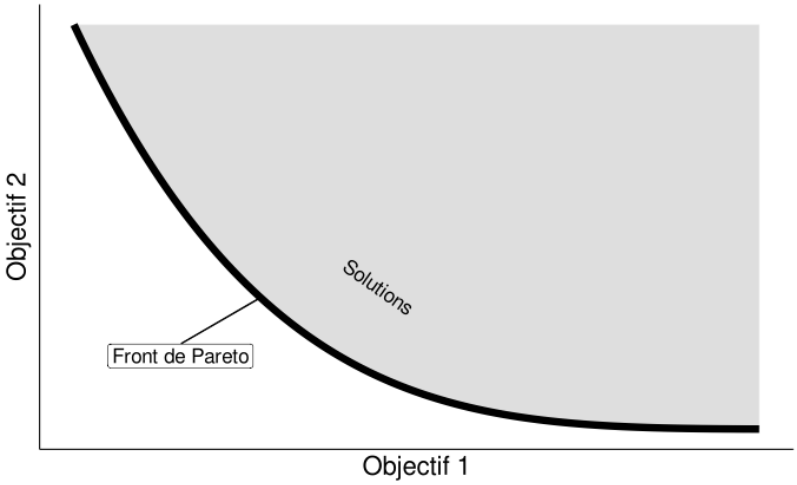


FIGURE 3.2 – Front de Pareto lors d'un problème de minimisation à 2 objectifs

pour répondre à nos objectifs est de construire le front de Pareto. Pour le construire, nous devons introduire le concept de dominance Pareto, noté $\prec$.

Soient 2 vecteurs objectifs, $z,\ z' \in Z$, correspondant aux solutions $x,\ x' \in S$. On dit alors que la solution x est dominée par x', noté $x \prec x'$, lorsque les deux conditions suivantes sont vérifiées :

$$\forall m \in \{1, \ldots, M\},\ z'_m \leq z_m \tag{3.22}$$
$$\exists m \in \{1, \ldots, M\},\ z'_m < z_m \tag{3.23}$$

L'équation 3.22, correspond au fait que la solution x' doit être au moins aussi adaptée que x pour toutes les fonctions objectifs. L'équation 3.23, correspond au fait qu'il existe au moins une fonction objectif f_m, pour laquelle la solution x' retourne une valeur plus faible que x. Nous expliquons ici la dominance Pareto au travers de la minimisation de nos fonctions objectifs, impliquant qu'une solution plus adaptée réduit l'ensemble de nos objectifs par rapport à une autre.

Cette dominance Pareto permet d'éliminer les solutions les moins adaptées à notre problème. L'ensemble des solutions qui ne sont pas dominées correspond au front de Pareto, comme illustré dans la figure 3.2. En s'appuyant uniquement sur le front de Pareto, il est cependant impossible de désigner une solution unique. En effet, le choix reste subjectif par rapport au compromis entre tous les objectifs que peut faire la personne qui prend la décision.

3.3.2 Cartes de contrôle

Nous venons d'introduire quelques concepts nécessaires à la bonne compréhension de la partie statique, nous allons à présent introduire quelques concepts clés de la méthode de détection du moment de réplication présentée dans la partie 6.2.2 du chapitre 6. En effet, pour détecter ce moment, nous utilisons les cartes de contrôle.

3.3.2.1 Introduction

Les cartes de contrôle, proposées par [Shewhart, 1931], font partie des outils de Contrôle Statistique de Procédés (SPC). C'est une méthode de Contrôle Qualité tel que défini par l'Organisation Internationale de Normalisation [iso, 2015].

> "La qualité de produits et services d'une organisation est déterminée par la capacité à satisfaire la clientèle et les impacts volontaires et involontaires sur les parties concernées"

Le but est de déterminer si un processus est dans un état de contrôle ou non. Cela signifie que la chaîne d'activités permettant de fournir un produit ou un service fournit celui-ci en conformité avec les exigences de la clientèle. Si, par exemple, nous fournissons un service Cloud avec un accord de niveau de service (SLA) qui spécifie que le temps de réponse sera inférieur à 15 secondes, alors un processus sous contrôle sera l'ensemble des activités qui permettront de retourner la donnée en moins de 15 secondes sinon le processus sera hors de contrôle.

Les cartes de contrôles sont surtout utilisées dans l'industrie et dans la recherche associée à la gestion de qualité. Elles ont récemment été utilisées en informatique pour surveiller la qualité des données [Hazen et al., 2014, Jones-Farmer et al., 2014] ou encore pour maîtriser la variabilité des processus [Kim et al., 2019].

3.3.2.2 Théorie des probabilités

Pour mieux comprendre les cartes de contrôles, il est important d'avoir quelques notions de probabilité.

La théorie des probabilités correspond à l'étude des phénomènes caractérisés par le hasard et l'incertitude. Dans le cadre d'une expérience aléatoire, comme un lancé de dé par exemple, l'ensemble des résultats ou **évènements** possibles est nommé l'**univers** noté Ω. Une variable aléatoire est la variable qui représente les valeurs possibles d'une expérience.

Par exemple, dans un lancé de dé, l'univers comprend les valeurs $\Omega = \{1, 2, 3, 4, 5, 6\}$, et si le dé est cubique et parfaitement équilibré alors la probabilité que la variable aléatoire X soit égale à un évènement de l'univers est :

$$\mathbb{P}(X = 1) = \frac{Card(\{1\})}{Card(\Omega)} = \frac{1}{6} \tag{3.24}$$

Lorsque l'univers est dénombrable, la théorie des probabilités est discrète, et la variable aléatoire correspond à un évènement de l'univers. Les lois de probabilité dans un contexte discret correspondent à la liste des évènements associés de leurs probabilités.

Si l'univers n'est plus dénombrable alors la théorie des probabilités est continue et la variable aléatoire correspond à un intervalle. Les lois de probabilité continues sont caractérisées par des densités de probabilité qui permettent de décrire la répartition des chances d'un évènement. L'intégrale de cette densité permet d'obtenir la fonction de répartition de cette loi.

Dans les deux cas, la masse totale, c'est-à-dire la somme des probabilités de tous les évènements est égale à 1 dans le cas discret. La fonction de répartition tend vers 1 quand la variable aléatoire X tend vers $+\infty$.

Il existe ainsi des lois de probabilité définies qui permettent de représenter différents problèmes et qui dépendent de paramètres.

La loi de Bernoulli est une loi de probabilité discrète qui correspond à une expérience à deux issues : $\Omega = \{0, 1\}$. Cette loi, notée $\mathcal{B}(p)$, dépend d'un paramètre $p \in [0, 1]$ qui mesure la probabilité de succès (assimilé à 1) :

$$\mathbb{P}(X = 1) = p \quad et \quad \mathbb{P}(X = 0) = 1 - \mathbb{P}(X = 1) = 1 - p \tag{3.25}$$

Une loi Binomiale est une loi qui correspond au nombre de succès au terme de n expériences de Bernoulli indépendantes de paramètre p. Cette loi, notée $\mathcal{B}(n, p)$, dépend d'un paramètre n et d'un paramètre p, correspondant respectivement au nombre d'expériences et à la probabilité de succès. Ainsi, la probabilité de k succès, avec $k \in \{0, \ldots, n\}$, est de :

$$\mathbb{P}(X = k) = \binom{n}{k} p^k (1 - p)^{n-k} \tag{3.26}$$

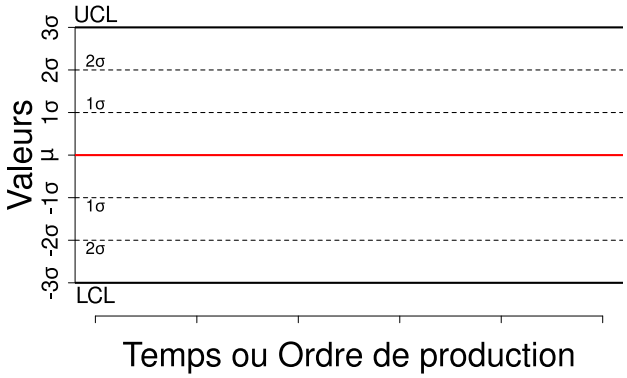


FIGURE 3.3 – Structure d'une carte de contrôle

Une dernière loi importante à mettre en avant est la loi Normale, qui est une loi continue. Elle dépend des paramètres μ et σ^2 qui correspondent respectivement à son espérance, c'est-à-dire le résultat moyen d'une expérience aléatoire, et sa variance. Elle est notée $\mathcal{N}(\mu, \sigma^2)$, sa densité de probabilité est :

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \tag{3.27}$$

A présent, nous allons décrire le fonctionnement des cartes de contrôle.

3.3.2.3 Fonctionnement

Les cartes de contrôle se basent sur des lois de probabilité dont on connaît ou estime les paramètres. Cela permet de construire une cible avec comme ligne centrale le paramètre ciblé. La connaissance de la distribution associée à la carte, permet de connaître la probabilité qu'un évènement arrive. Des limites de contrôle sont mises en place autour de la cible qui représentent les limites entre lesquelles un processus est considéré sous contrôle. La limite de contrôle supérieure est notée *Upper Limit Control* (UCL) et la limite inférieure, *Lower Limit Control* (LCL).

En se basant sur la théorie du 6-Sigma décrite par [Montgomery and Woodall, 2008], en supposant une loi normale, un évènement qui arrive au-delà des 6σ a une probabilité de 2 par million d'arriver. De même, la probabilité qu'un évènement dépasse les 3σ est de 0.27%. Les limites de contrôle sont ainsi situées à 3σ autour de la cible pour avoir une probabilité proche de 0.1% de chaque coté. Au sein de ces limites de contrôles sont définies 3 zones de chaque coté de la ligne centrale qui correspondent à chaque sigma : zone $0-1\sigma$, zone $1-2\sigma$ et la zone $2-3\sigma$.

Une carte de contrôle se présente ainsi comme dans la figure 3.3.

Après la création de la carte de contrôle, des règles sont mises en place qui permettent de lever des alertes. Ces règles, dans un premier temps proposées par [Western Electric, 1956], ont été ensuite étendues par [Nelson, 1984].

Règle 1 : 1 point au-delà de la zone 3σ
Règle 2 : 9 points ou plus qui sont du même coté de la moyenne
Règle 3 : 6 points ou plus qui augmentent (ou diminuent) continuellement
Règle 4 : 14 points ou plus en continu dont la direction alterne
Règle 5 : Au moins 2 points sur 3 qui sont au-delà de 2σ dans la même direction
Règle 6 : Au moins 4 points sur 5 qui sont au-delà de 1σ dans la même direction
Règle 7 : 15 points en continu qui sont dans la zone de 1σ quelque soit le côté par rapport à la moyenne
Règle 8 : 8 points en continu qui ne sont pas dans la zone de 1σ quelque soit le côté par rapport à la moyenne

Ces règles n'ont pas pour objectif d'être mises en place toutes en même temps, car cela augmente la possibilité de fausses alertes.

3.3.2.4 Types de Cartes

Comme le montre [Western Electric, 1956], plusieurs types de cartes de contrôle sont proposés. Le choix de carte se fait selon les processus à garder sous contrôle, la disponibilité des données, les paramètres à prendre en compte et donc les lois de probabilité auxquels elles se rapportent. Il est possible de regrouper les cartes de contrôle en 3 catégories.

Shewhart : Les cartes de Shewhart [Shewhart, 1931] correspondent à des cartes cherchant à garder sous contrôle une caractéristique moyenne, par exemple la largeur d'une porte à la sortie d'une machine. Ces cartes fonctionnent par paire avec : une carte de distribution pour contrôler les variations, et une carte de moyenne. Cette catégorie de carte de Shewhart inclue les cartes individuelles de Shewhart, les cartes $\overline{X}$ et S (pour l'écart-type) ainsi que les cartes $\overline{X}$ et R (pour l'étendue).

Attributs : Les cartes d'attributs correspondent à la fréquence d'apparitions d'un évènement. Ces cartes se divisent en 2 groupes, qui correspondent à 2 distributions différentes. (i) Les cartes qui comptent le nombre de produits défectueux suivent une loi Binomiale. Si on tient compte de la proportion de produits défectueux, c'est une carte p ; si on considère le nombre de produits défectueux, c'est une carte np. Et (ii) les cartes qui comptent le nombre de défauts sur un produit suivant une loi de Poisson. De même, les cartes qui s'appuient sur le pourcentage de défauts sont nommées les cartes u, et les cartes qui considèrent le nombre de défauts sont nommées les cartes c.

Autres : Les catégories décrites précédemment sont les cartes les plus fréquemment utilisées. Il existe cependant beaucoup d'autres cartes répondant à des problématiques différentes. Par exemple, les cartes *Exponentially Weigthed Moving Average* ou *EWMA* [Neubauer, 1997] qui permettent de prendre en compte l'historique, dans la détermination d'un point hors contrôle. Ou encore, les cartes multidimensionnelles [Zhang et al., 2010], qui permettent de répondre au fait que si dans un processus le nombre de carte devient important, alors le nombre de fausses alertes devient aussi plus important.

3.4 Conclusion

Dans ce chapitre, nous avons présenté les principaux modèles et outils utilisés dans la proposition de stratégie de réplication de données.

Les modèles de consommation énergétique et de dépense représentent une base sur laquelle s'appuieront les fonctions objectifs présentées dans le chapitre 5. C'est dans ce chapitre que seront utilisés les concepts d'optimisation multi-objectif.

Les cartes de contrôle sont utilisées dans le chapitre 6, afin de détecter le moment de la réplication.

Dans le chapitre suivant, nous allons faire une validation des modèles présentés dans ce chapitre. Et nous allons présenter l'architecture Cloud que nous considérons, les paramètres économiques et énergétiques utilisés et la mise en place de la simulation.

Notations	Description	Unité
ϕ	Matrice représentant le placement des données sur les nœuds	
B^j_{disk}	Bande-passante du disque du nœud j	Go/s
B^j_{NIC}	Bande-passante de la carte réseau du nœud j	Go/s
B^j_{RAM}	Bande-passante de la mémoire du nœud j	Go/s
B^p_{SW}	Bande-passante du commutateur réseau p	Go/s
cp^j_{disk}	Capacité de stockage du disque dur du nœud j	Go
cp^j_{RAM}	Capacité de stockage de la mémoire du nœud j	Go
E^{CPU}_{act}	Consommation énergétique du processeur en activité	J
E^{disk}_{act}	Consommation énergétique du disque en activité	J
E^{NIC}_{act}	Consommation énergétique de la carte réseau en activité	J
E^{SW}_{act}	Consommation énergétique du commutateur en activité	J
E^{CPU}_{idle}	Consommation énergétique du processeur en inactivité	J
E^{disk}_{idle}	Consommation énergétique du disque en inactivité	J
E^{NIC}_{idle}	Consommation énergétique de la carte réseau en inactivité	J
E^{RAM}_{idle}	Consommation énergétique de la mémoire en inactivité	J
E^{SW}_{idle}	Consommation énergétique du commutateur en inactivité	J
E^{RAM}_{read}	Consommation énergétique de la mémoire en lecture	J
E^{RAM}_{write}	Consommation énergétique de la mémoire en écriture d'une données	J
EC_{Exec}	Consommation énergétique de l'exécution d'une tâche	J
EC_{Read}	Consommation énergétique de la lecture d'une donnée	J
EC_{Static}	Consommation énergétique globale d'un nœud	J
EC_{Write}	Consommation énergétique de l'écriture d'une donnée	J
EX_{Exec}	Dépense associée à l'exécution d'une donnée	$
$EX_{Network}$	Dépense associée au transfert d'une donnée	$
$EX_{SLO\_violation}$	Dépense associée aux violations d'objectif de niveau de service	$
$EX_{Storage}$	Dépense associée au stockage de la donnée	$
F	Ensemble des fichiers présents dans l'architecture	
f_i	Un fichier i avec $f_i \in F$, $1 \leq i \leq z$	
$f(t_k)$	Ensemble des fichiers nécessaires à la tâche k	
fr_j	Fréquence du processeur du nœud j	Hz
$lvl(sw_p)$	Niveau du commutateur p, niveaux : {Région, Centre de données}	
N	Ensemble des nœuds de l'architecture	
n_j	Un nœud j avec $n_j \in N$, $1 \leq j \leq m$	
$nodes(sw_p)$	Ensemble des nœuds connectés au commutateur p	

TABLE 3.1 – Résumé des notations introduites dans le chapitre – Partie 1

Notations	Description	Unité
$P^{fr_j}_{CPU_{idle}}$	Puissance en inactivité du processeur à la fréquence fr_j du nœud j	W
$P^{fr_j}_{CPU_{max}}$	Puissance maximale du processeur à la fréquence fr_j du nœud j	W
$P^{j}_{disk_{active}}$	Puissance du disque dur en activité du nœud j	W
$P^{j}_{disk_{idle}}$	Puissance du disque dur en inactivité du nœud j	W
$P^{j}_{disk_{seek}}$	Puissance du disque dur en recherche du nœud j	W
$P^{j}_{NIC_{active}}$	Puissance de la carte réseau en activité du nœud j	W
$P^{j}_{NIC_{idle}}$	Puissance de la carte réseau en inactivité du nœud j	W
$P^{j}_{RAM_{idle}}$	Puissance de la mémoire en inactivité du nœud j	W
$P^{j}_{RAM_{read}}$	Puissance de la mémoire en lecture du nœud j	W
$P^{j}_{RAM_{write}}$	Puissance de la mémoire en écriture du nœud j	W
$P^{p}_{SW_{active}}$	Puissance du commutateur p en activité	W
$P^{p}_{SW_{idle}}$	Puissance du commutateur p en inactivité	W
$Port^{p}$	Nombre de port du commutateur p	
$Pr^{n_j}_{Exec}$	Prix de l'exécution d'une tâche sur le nœud j	\$/Instruction
$Pr_{Network}$	Prix du transfert entre 2 nœuds	\$/Go
$Pr^{n_j}_{Storage}$	Prix du stockage sur un nœud	\$/Go par s
$Prop_{Refund}(level)$	Pourcentage de pénalité lorsque l'objectif de niveau de service n'est pas atteint	%
$s(f_i)$	Taille du fichier i	Go
$s(t_k)$	Nombre d'instructions de la tâche t_k	
SW	Ensemble des commutateurs réseaux de l'architecture	
sw_q	Un commutateur réseau avec $sw_p \in SW$, $1 \leq p \leq q$	
T	Ensemble des tâches exécutées dans l'architecture	
t_k	Une tâche k avec $t_k \in T$, $1 \leq k \leq y$	
t^{j}_{seek}	Temps moyen de recherche d'une donnée sur le disque du nœud j	s
T^{j}_{tot}	Temps total d'utilisation du nœud j	s
tps_{SLO}	Objectif de niveau de service en terme de temps de réponse	s
$tps(t_k)$	Temps total d'exécution de la tâche k	s
u_j	Pourcentage d'utilisation du processeur du nœud j	%

Table 3.2 – Résumé des notations introduites dans le chapitre – Partie 2

Chapitre 4

Architecture et Environnement expérimental

Sommaire

4.1 Introduction

Dans le chapitre précédent, nous avons introduit les modèles de consommation énergétique et de dépense utilisés dans la suite du manuscrit en plus de quelques outils qui seront utilisés dans la stratégie de réplication de données.

Dans ce chapitre, nous introduisons l'architecture considérée dans la suite de ce manuscrit. Notre stratégie est implémentée côté fournisseur s'appuyant sur des serveurs présents dans différentes régions du monde. Ces serveurs sont dans des centres de données dont les caractéristiques diffèrent d'une région à l'autre. Les informations des composants de ces centres proviennent de la plateforme expérimentale Grid5000 [Balouek et al., 2013].

C'est à l'aide de cette plateforme expérimentale que nous faisons la validation de nos modèles de consommation énergétique présentés précédemment.

Sur ces bases, nous présenterons l'environnement expérimental qui permettra l'évaluation des performances de la stratégie de réplication de données proposée.

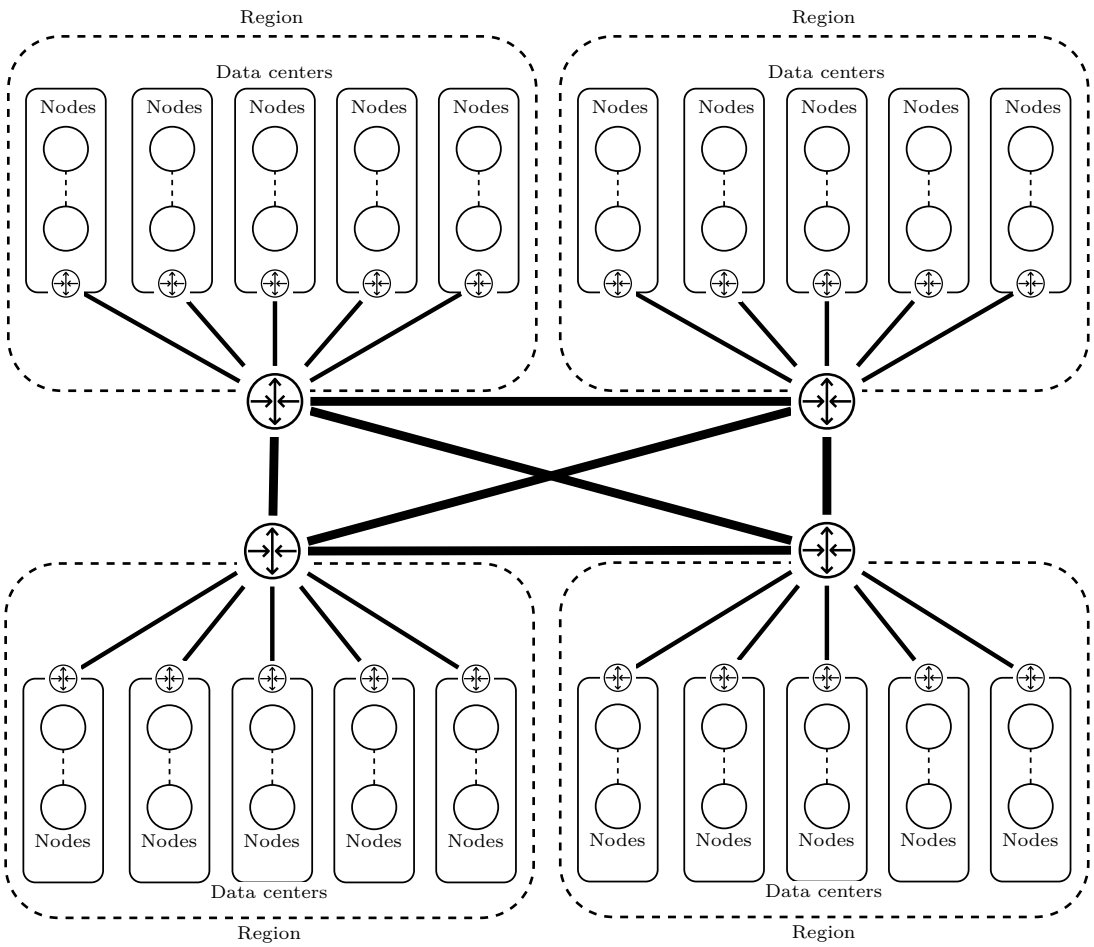


FIGURE 4.1 – Représentation de l'architecture considérée

4.2 Architecture

Nous allons dans cette partie présenter l'architecture considérée lors de la conception de notre stratégie de réplication de données et que nous avons également simulée lors de nos expériences.

4.2.1 Topologie

Nous faisons l'hypothèse d'être un fournisseur de service Cloud international. Ce fournisseur est présent dans quatre régions, et chaque région possède 5 centres de données. Chaque centre de données est composé de 64 nœuds de calcul, permettant d'exécuter des tâches et de stocker des données.

Les nœuds sont connectés entre eux de manière hiérarchique. Les nœuds d'un même centre de données sont connectés au commutateur du centre de données, et les centres de données sont connectés aux commutateurs de leur région. Cette topologie est représentée sur la figure 4.1.

Au niveau du réseau, nous nous basons sur les travaux de [Boru et al., 2015] pour la bande-passante, et des documents techniques de Wikipédia [1] pour la latence. Ainsi, entre les régions, la bande-passante et la latence sont de 100Gbits/s et 160ms respectivement. Dans une région et dans un centre de données, la bande-passante est de 10Gbits/s et les latences sont de 30ms et 1μs respectivement.

1. https://wikitech.wikimedia.org/wiki/Network_design (08/28/2020)

L'architecture adoptée est résumée dans le tableau 4.1.

Paramètres	Valeurs
Nombre de nœuds par centre	64
Nombre de centres par région	5
Nombre de régions par Cloud	4
Bande-passante entre régions	100 Gbit/s
Bande-passante dans une région	10 Gbit/s
Bande-passante dans un centre	10 Gbit/s
Latence entre régions	160 ms
Latence dans une région	30 ms
Latence dans un centre	1 μs

TABLE 4.1 – Paramètres d'architecture

4.2.2 Configuration matérielle

Dans la proposition de la stratégie de réplication de données, nous nous appuyons sur l'hétérogénéité entre les nœuds de différents centres de données. Pour cela, tous les centres de données d'une région sont associés à un profil de nœuds. Ce profil correspond à un choix de serveur avec un processeur, de la mémoire, un disque dur et une carte réseau.

Ces profils correspondent à du matériel provenant de différents clusters de Grid5000. Le **profil 1** correspond au cluster *Paranoïa*[2] de Rennes qui est un cluster de 2019. Le **profil 2** correspond au cluster *Gros*[3] de Nancy qui est un cluster de 2014. Le **profil 3** correspond au cluster *Dahu*[4] de Grenoble qui est un cluster de 2018. Pour des raisons de temps, nous n'avons pas pu intégrer les commutateurs réseaux de Grid5000, nous nous sommes donc basés sur les commutateurs que nous avions déjà implémenté auparavant. Un résumé des composants correspondant à chaque profil est présenté dans le tableau 4.2.

Profils	Composant	Détails
Profil 1 : Paranoïa	Processeur	Intel Xeon E5-2660 v2 – 10 cœurs/CPU et 2 CPUs/nœud
	Mémoire	128 GiB
	Disque dur	600 GB HDD SAS Western Digital
	Carte réseau	Intel 82599ES 10Gb SFI/SFP+
	Commutateur	HPE Altoline 6960 Switch Series
Profil 2 : Gros	Processeur	Intel Xeon Gold 5220 – 18 cœurs/CPU et 1 CPU/nœud
	Mémoire	96 GiB
	Disque dur	960 GB SSD SATA Micron
	Carte réseau	25 Gbps, Mellanox Technologies
	Commutateur	Cisco Nexus 9300-EX
Profil 3 : Dahu	Processeur	Intel Xeon Gold 6130 – 16 cœurs/CPU et 2 CPUs/nœud
	Mémoire	192 GiB
	Disque dur	4.0 TB HDD SATA Seagate
	Carte réseau	10 Gbps, Intel Ethernet Controller X710
	Commutateur	HPE Altoline 6960 Switch Series

TABLE 4.2 – Configuration matérielle de chaque profil

2. https://www.grid5000.fr/w/Rennes:Hardware#paranoia
3. https://www.grid5000.fr/w/Nancy:Hardware#gros
4. https://www.grid5000.fr/w/Grenoble:Hardware#dahu

4.3 Validation des modèles énergétiques

Dans la partie 3.2 du chapitre 3, nous avions introduit les modèles de consommation énergétique que nous utilisons tout au long de ce manuscrit. Ces modèles sont issus de la littérature et nous voulions valider ces modèles dans le cadre de nos travaux.

4.3.1 Configuration de validation

Pour mettre en place cette validation, nous nous sommes appuyés sur la plateforme Grid5000 qui est une plateforme d'expérimentation. Elle correspond à une grille de calcul, avec des clusters présents sur plusieurs sites. L'ensemble des sites sont interconnectés à l'aide du réseau Renater, et sont accessibles de n'importe où.

Sur cette plateforme, certains clusters sont équipés de wattmètre au niveau de l'unité d'approvisionnement d'électricité, dont les données sont accessibles via une API. Pour des raisons de disponibilité et de simplicité, nous nous sommes concentrés sur le cluster *Gros* de Nancy qui possède un wattmètre par nœud. De plus, ce cluster est constitué de 124 nœuds permettant d'être fréquemment disponible. Une description des composants des nœuds de Gros est proposée sur le tableau 4.3 s'appuyant sur les informations fournies par Grid5000 [5].

Composant	description
Processeur	Intel Xeon Gold 5220 18 cœurs/CPU et 1CPU/nœud
Mémoire	96 GiB
Disque dur	480 GB SSD SATA Micron 960 GB SSD SATA Micron
Carte réseau	25 Gbps, Mellanox Technologies 25 Gbps, Mellanox Technologies

Table 4.3 – Composant de chaque nœud de gros

Dans le cadre de nos expérimentations, nous avons stressé chaque composant de manière aussi indépendante que possible. Pour cela, nous nous sommes appuyés sur le benchmark utilisé dans [Da Costa et al., 2020], permettant de stresser le CPU, la mémoire et la carte réseau. Pour stresser le disque, nous avons utilisé le benchmark Iozone [6].

L'outil proposé par [Da Costa, 2021], permet de lancer ces benchmarks et de récupérer les informations de performance et de consommation tout en respectant les temporalités pour faire coïncider les actions avec les données récupérées. Avant d'exécuter les stress, nous avons au préalable désactivé les modes inactif et turbo du CPU.

Cette partie se divise donc en une estimation de la consommation en **inactivité**, puis une validation des modèles de puissance du **processeur**, de la **mémoire**, de la **carte réseaux** et du **disque**.

4.3.2 Résultats

4.3.2.1 Inactivité

Dans un premier temps, nous avons lancé un ensemble d'expériences qui gardent le nœud en inactivité avec uniquement les tâches de fond. Pour cela, nous avons lancé la commande *sleep* pendant 10 minutes. Nous obtenons ainsi, le résultat suivant présenté dans la figure 4.2.

La puissance statique nécessaire pour garder le nœud en inactivité est donc de 130.88 Watt en moyenne avec un écart-type de 0.93 W.

5. https://www.grid5000.fr/w/Nancy:Hardware#gros
6. http://iozone.org/

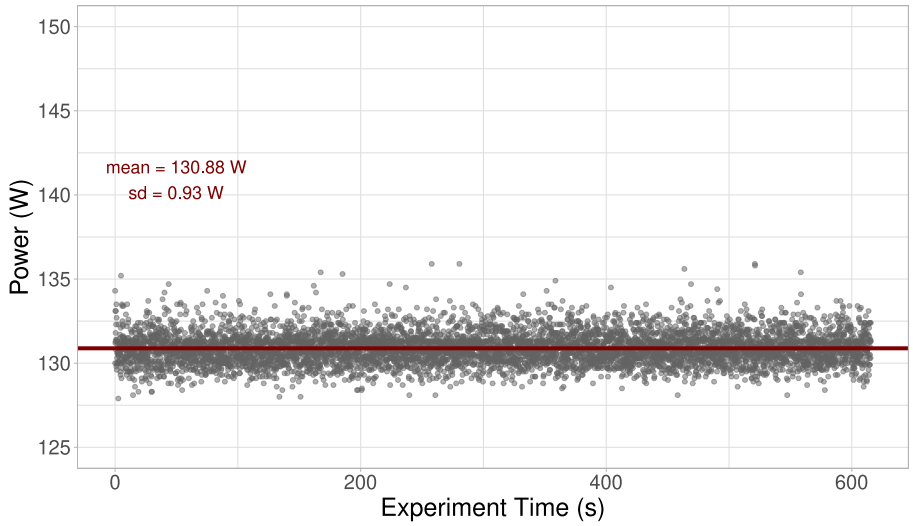


FIGURE 4.2 – Puissance en inactivité

Par la suite, nous considérons que la puissance statique est de 130.88 W, pour différencier l'impact de chaque composant.

4.3.2.2 Processeur

Le processeur est le composant qui consomme le plus de puissance, et donc celui qui a le plus d'impact sur la consommation énergétique. Pour valider notre modèle, nous nous sommes appuyés sur un stress test qui génère des valeurs aléatoires : *Rand*. Le test va donc lancer ce stress de manière incrémentale sur chaque cœur du CPU. Les résultats sont présentés sur la figure 4.3.

Dans cette figure, nous avons comparé le modèle linéaire de puissance du processeur proposé par [Beloglazov et al., 2012] :

$$P_{Active}(u) = (P^{fr_j}_{CPU_{max}} - P^{fr_j}_{CPU_{idle}}) * u + P^{fr_j}_{CPU_{idle}} \tag{4.1a}$$

avec le modèle non-linéaire proposé par [Fan et al., 2007] :

$$P_{Active}(u) = (P^{fr_j}_{CPU_{max}} - P^{fr_j}_{CPU_{idle}}) * (2u_j - u_j^r) + P^{fr_j}_{CPU_{idle}} \tag{4.2a}$$
$$Avec : \ r = 1.4$$

Sur le résultat, nous voyons que le modèle non linéaire est plus performant que le modèle linéaire. Cette performance est décrite sur la figure avec le calcul du R^2, avec une valeur de 0.9906 pour le modèle non linéaire, contre 0.9855 pour le modèle linéaire. Un autre indicateur de performance peut être mis en avant : la moyenne du carré des erreurs ou *MSE*. Contrairement au R^2, plus le *MSE* est faible, meilleur est le modèle. Ici, le *MSE* pour le modèle linéaire est de 3.9 contre 0.9 pour le modèle non linéaire.

4.3.2.3 Mémoire

Dans la modélisation de la mémoire, nous avions supposé qu'il y a une différence significative entre les états d'écritures et de lectures sur la mémoire RAM. Pour cela, un stress de la mémoire RAM est effectué. Il permet de forcer des écritures sur la mémoire durant un certain temps, puis de forcer une lecture durant ce même temps.

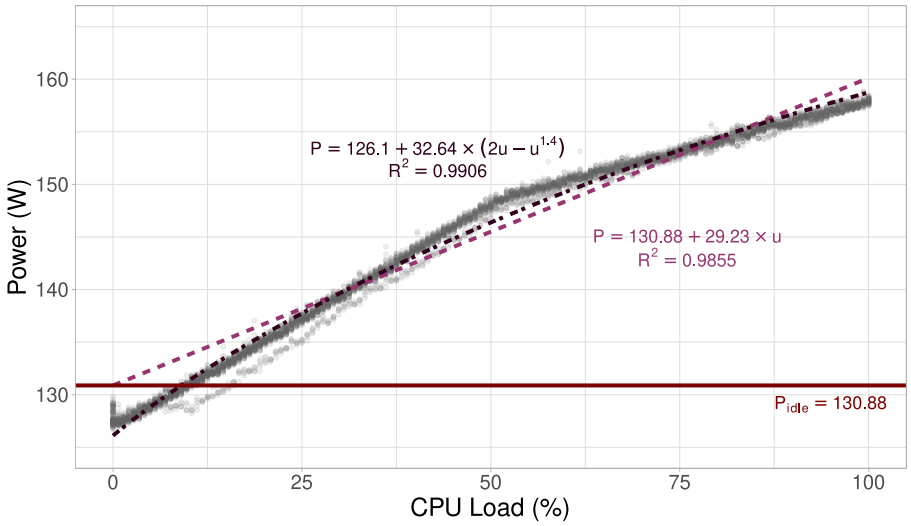


FIGURE 4.3 – Puissance par pourcentage d'utilisation du CPU

L'expérience consistait donc en une alternance entre lecture et écriture de 30 secondes chacune. Cela nous permet d'obtenir le résultat présenté sur la figure 4.4.

Elle montre que le stress de la mémoire RAM a un impact significatif sur la puissance moyenne du nœud. En effet, la puissance moyenne de ce stress est de 133.5W contre 130.9W lorsque le nœud est inactif. De plus, le type d'action effectuée sur la mémoire RAM a un impact sur la puissance du nœud. Cet impact est estimé sur le nœud à 1.4W lorsque la RAM est en lecture et 4.2W lorsque la RAM est en écriture.

4.3.2.4 Carte réseau

Pour valider l'estimation de l'impact de la carte réseau sur la consommation du nœud, le benchmark stresse la carte réseau en *Upldoad* ou *Download* à l'aide d'un second nœud récepteur. Ce benchmark permet aussi de moduler la bande-passante utilisée pour le stress. Ici, le nœud récepteur est un nœud du même cluster.

L'expérience consiste en une alternance entre *Upload* et *Download* durant 30 secondes chacun pour une bande-passante de 200, 400 et 1000 Mo/s. Les résultats sont présentés sur la figure 4.5.

Elle montre que l'utilisation de la carte réseau a un impact significatif sur la consommation du nœud avec une augmentation de la puissance de 2.8W en moyenne. Dans le cadre de notre modèle, nous avions supposé que la puissance est principalement associée à son activation, et non à l'utilisation de la bande-passante, ni à l'action que réalise la carte réseau. Ainsi, nous avons estimé que la différence de consommation entre l'action de la carte réseau est de l'ordre de 0.05W, et chaque giga-octet de bande-passante augmente la consommation de 0.25W.

4.3.2.5 Disque dur

Pour estimer la puissance du disque dur, nous avons utilisé l'outil de benchmark *Iozone*. Cet outil a le désavantage de devoir générer les fichiers dans la mémoire RAM, avant d'écrire le fichier sur le disque. Il est difficile de séparer les moments de préparation du fichier, du moment d'écriture sur le disque. Par ailleurs, il n'est pas possible de lancer une expérience uniquement basée sur la lecture seule. Il faut obligatoirement faire une expérience d'écriture-lecture pour que cela fonctionne. Le but de cette validation est donc de vérifier que le disque dur à un impact significatif sur la consommation énergétique lors de la gestion de fichiers.

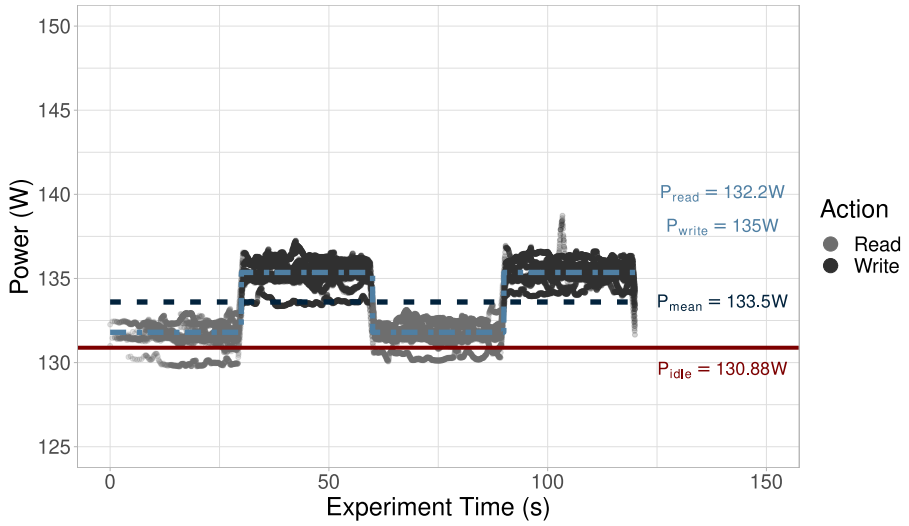


FIGURE 4.4 – Puissance de la Mémoire RAM

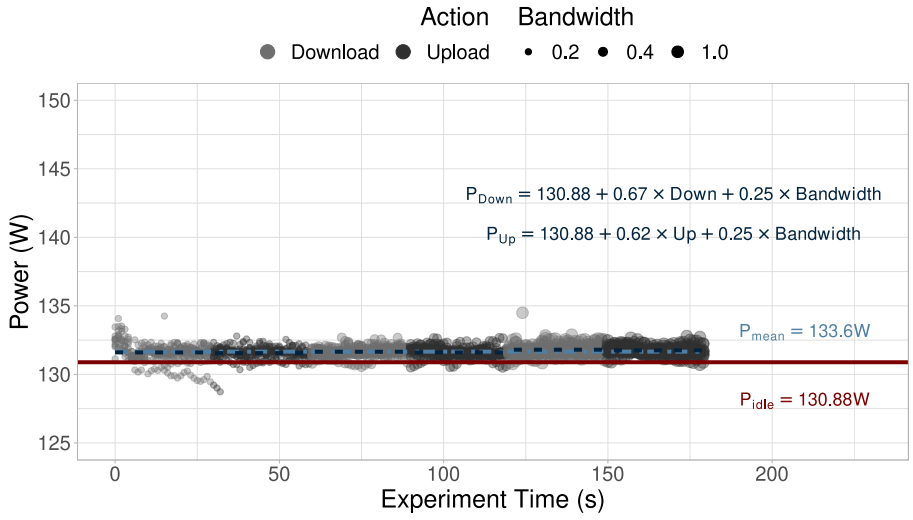


FIGURE 4.5 – Puissance de la Carte Réseau selon l'action et la bande-passante (Go/s)

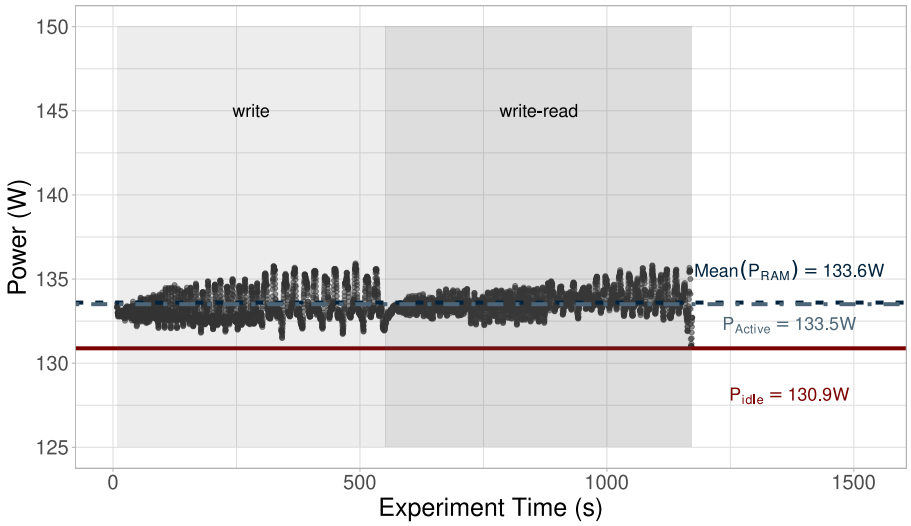


FIGURE 4.6 – Puissance du stress du disque dur

L'expérience consiste à générer des fichiers de tailles allant de 512Mo à 16Go (en doublant la valeur) transférés aux travers de blocs de taille allant de 512Ko à 8Mo (en doublant la valeur). Le résultat est présenté sur la figure 4.6.

Sur cette expérience, nous voyons sur la figure, que l'impact du disque dur sur la puissance se mélange avec la puissance nécessaire à la création du fichier sur la mémoire RAM. Cette absence d'impact que l'on pourrait associer à l'utilisation du disque, pourrait être expliquée par le fait que le disque présent sur le nœud est un SSD, qui n'a pas les mêmes besoins en puissance qu'un HDD devant faire tourner un moteur par exemple.

De plus, probablement qu'avec un outil plus orienté sur le stress du disque uniquement, nous aurions pu extraire la puissance nécessaire au disque dur uniquement.

4.3.2.6 Commutateur réseau

Dans le cadre de nos travaux, nous n'avons pas eu l'occasion de valider le modèle de consommation énergétique de l'utilisation des commutateurs réseaux. Différents travaux mettent en avant l'impact ou non de certains paramètres.

Les auteurs de [Sohan et al., 2010] montrent l'impact du nombre de liens actifs sur une carte de réseaux multi-ports. Ils mettent en avant le fait que plus une carte réseau a de ports, plus la consommation énergétique de la carte est élevée. De plus, un résultat intéressant est que sur les cartes multi-ports, augmenter le nombre de liens actifs augmente la consommation statique de la carte réseau.

Les auteurs de [Hlavacs et al., 2009] évaluent l'impact de l'utilisation réseau sur ces cartes réseaux. Ils montrent que dans la plupart des cas, l'impact est statistiquement significatif, mais faible. En effet, les valeurs associées à cette augmentation sont de l'ordre de $0.01W$ par $\log BW$ avec BW en kbits/s.

Nous nous appuyons sur ces résultats pour implémenter les modèles de consommation énergétique au sein du simulateur. Ce dernier est introduit dans la partie suivante qui présente l'environnement expérimental utilisé dans les chapitres suivants.

4.4 Environnement expérimental

Dans cette dernière partie, nous présentons l'environnement expérimental que nous avons considéré lors de l'évaluation des performances de la stratégie de réplication de données proposée. Cette évaluation est présente dans le placement initial (partie 5.4 du chapitre 5) et dans la réplication dynamique (partie 6.5 du chapitre 6) de la stratégie. Nous commençons par introduire le simulateur ainsi que les modifications et ajouts effectués. Puis nous présenterons les paramètres énergétiques et économiques pour finir avec les charges de travail considérées.

4.4.1 Simulateur

Le simulateur utilisé dans l'évaluation des performances est CloudSim, développé par [Calheiros et al., 2011] puis étendu par [Tos et al., 2017] pour mettre en place un modèle économique et une architecture large échelle.

Nous avons intégré à cette extension, la prise en compte de la consommation énergétique à l'aide des modèles proposés dans le chapitre 3. À cela s'est ajoutée une hétérogénéité entre les centres de données en terme de caractéristiques matérielles, énergétiques et économiques.

Nous avons aussi mis en place la technique du *Sleep State* permettant de mettre les nœuds dans un état de veille qui consomme moins, mais qui nécessite un temps de réveil.

Nous avons implémenté un algorithme de placement de tâches naïf qui a connaissance de l'état de toutes les tâches présentes sur les nœuds de son centre de données. Il favorise le placement des tâches sur les nœuds déjà allumés selon le temps de transfert des données et le temps de calcul de la tâche. La décision de placer la tâche sur un nœud en veille dépend du fait que les SLOs ne seront pas satisfaits, quelque soit le nœud actif sur lequel la tâche sera placée. Si un nœud est dans un état d'activité mais n'exécute plus de tâche pendant un certain temps, alors il se met en état de veille. Dans ce contexte, nous supposons qu'un nœud qui stocke des données ne peut être mis en veille pour répondre aux requêtes le plus rapidement possible.

Ce placement de tâches est loin d'être optimal surtout en termes de temps de calcul et de transfert d'informations, car cela implique que le planificateur connaisse l'état de l'ensemble du système. D'autres algorithmes de placement de tâches ont été étudiés pour prendre en compte la consommation énergétique [Roukh et al., 2016] et les intermittences associées à l'utilisation d'énergies renouvelables [Grange et al., 2018, De Courchelle et al., 2019].

4.4.2 Paramètres considérés

Dans la partie 4.2, nous avons introduit la topologie du fournisseur Cloud. Nous allons à présent introduire les profils de (i) consommation énergétique et de (ii) coûts associés aux centres de données de chaque région, ainsi que (iii) les objectifs de niveau de service (*SLO*) que le fournisseur doit atteindre.

Énergie Les paramètres de consommation énergétique sont issus des composants des profils présentés dans la section 4.2. À l'aide des fiches techniques correspondant à ces composants, nous utilisons les valeurs de puissance présentées sur le tableau 4.4. Ces valeurs sont les entrées des modèles présentés dans le chapitre précédent.

Coûts Enfin, un dernier ensemble de paramètres utiles à la simulation, sont les paramètres de coûts. Ici, nous nous basons sur les prix, proposés par Google [7] dans différentes régions, auxquels on applique une marge de 20%. Google ayant des centres de données dans différentes régions du monde, le prix de la location de ressources dépend de la région. Les coûts utilisés correspondent aux coûts des régions considérées dans le Cloud de Google. Cependant, pour le revenu, nous considérons que le fournisseur a un prix unique par tâche pour l'ensemble des régions afin de faciliter la mise

7. https://cloud.google.com/ (28/02/2020)

Puissance	Profil 1	Profil 2	Profil 3
CPU Inactif	152 W	100 W	200 W
CPU Actif	191.7 W	132.64 W	265.28 W
Nombre de cœurs	20	18	32
Capacité de calcul (MIPS/cœur)	2200	2200	2100
RAM Inactif	22 W	4.9 W	4.9 W
RAM Lecture	27.25 W	6.8 W	6.8 W
RAM Écriture	27.25 W	8.6 W	8.6 W
RAM Bande-Passante	8528 MB/s	8528 MB/s	8528 MB/s
HDD Inactif	4.2 W	3 W	5 W
HDD Lecture	5.45 W	4.45 W	9.7 W
HDD Écriture	5.45 W	4.45 W	9.7 W
HDD Seek	4 ms	0.004 ms	4 ms
HDD Bande-Passante	750 MB/s	750 MB/s	750 MB/s
NIC Inactif	2 W	3.8 W	1.1 W
NIC Actif	5.1 W	7.4 W	2 W
NIC Bande-Passante	1250 MB/s	3125 MB/s	1250 MB/s
Switch DC Inactif	267 W	210 W	267 W
Switch DC Actif	315 W	470 W	315 W
Switch DC Nombre de Ports	32	48	32
Switch Region Inactif	534 W	420 W	534 W
Switch Region Actif	630 W	940 W	630 W
Switch Region Nombre de Ports	64	96	64

TABLE 4.4 – Paramètres de Puissance

en place des coûts de violations. Les profils correspondent respectivement aux régions : (i) Londres, (ii) Taiwan, (iii) Zurich et (iv) Hong-Kong. Ces régions ont été tirées au hasard sans remise parmi l'ensemble des régions disponibles sur Google. Un résumé des paramètres de coûts est présenté sur le tableau 4.5

Objectif de niveau de service Nous avons introduit dans la partie économique les paliers de coûts des violations d'objectif de niveau de service. Dans notre contexte, on suppose que l'objectif de niveau de service correspond à un temps de réponse de 15 secondes en dessous duquel la requête doit être exécutée. Ce temps de réponse provient de [Nah, 2004], qui montre qu'une attente de plus de 15 secondes sur une page web, sans réaction (sans barre de chargement, par exemple), fait partir 25% de le clientèle.

4.4.3 Charge de travail

Enfin, pour comparer la stratégie de réplication proposée avec d'autres stratégies de la littérature, 2 types d'expérimentation sont mises en place.

Court terme (XP1) Une première expérimentation à court terme, qui considère une expérience de 4h avec 150k tâches à réaliser. Pour la charge de cette première expérience, nous nous basons sur [Moyer et al., 2015] qui montre que lorsqu'il y a un post sur un réseau social avec un lien Wikipédia, il y a une augmentation de l'intérêt pour le sujet, et augmente ainsi le nombre de vues sur la page Wikipédia. Puis l'intérêt disparaît petit à petit, ce qui réduit le nombre de vues sur la page. [Lerman and Ghosh, 2010] montre que cette baisse d'intérêt diminue à différentes vitesses. Nous supposons ainsi une charge de fond correspondant à un quart des tâches à l'aide d'une distribution uniforme sur l'ensemble de l'expérience. Le pic de charge est modélisé par une loi Gamma avec les paramètres $\alpha = 4$ et $\beta = 600$ répartissant les trois-quarts des tâches restant sur la dernière moitié de l'expérience. Cette charge est représentée dans la figure 4.7.

Coûts	Londres	Taïwan	Zurich	Hong-Kong
Exécution ($/MI)	$4.1*10^{-9}$	$3.7*10^{-9}$	$4.5*10^{-9}$	$4.5*10^{-9}$
Stockage ($/GB)	$4*10^{-8}$	$3.4*10^{-8}$	$7.8*10^{-8}$	$2.7*10^{-8}$
Coûts de transfert				
Intra-DC ($/GB)	$7.8*10^{-4}$	$7.8*10^{-4}$	$7.8*10^{-4}$	$7.8*10^{-4}$
Intra-région ($/GB)	$7.8*10^{-3}$	$7.8*10^{-3}$	$7.8*10^{-3}$	$7.8*10^{-3}$
Entre régions ($/GB)	0.094	0.094	0.094	0.094

Violations	% de violations	Coûts
	<1%	10% du revenu
	[1%,5%[	25% du revenu
	>=5%	100% du revenu

Revenu	$8.1*10^{-3}$$/Tâche

TABLE 4.5 – Paramètres de coûts

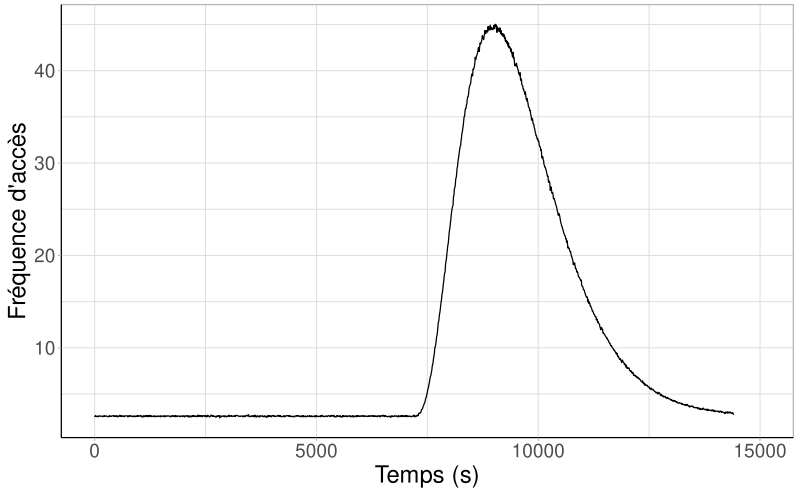


FIGURE 4.7 – Charge de travail de la 1ère expérience (XP1)
Tranche de 10 secondes

Long terme (XP2) La seconde expérimentation s'appuie sur [Yoshida et al., 2015] qui montre que les recherches sur google sont liées à des recherches Wikipédia. Ainsi, la corrélation des données par heure de google trend et les données par jour de Wikipédia nous a permis de représenter cette charge avec des fréquences d'accès en figure 4.8, que l'on peut adapter au nombre de tâches. Cette charge correspond à la semaine d'ouverture des vaccinations de la COVID-19 en France à l'ensemble de la population le 29 mai 2021. Cette ouverture permet de générer un pic de requêtes et donc de modéliser plus fidèlement de fortes charges. La seconde expérimentation est une expérience de 4 jours avec 2 millions de tâches.

Un résumé des paramètres des expériences est présenté dans le tableau 4.6. Dans ces paramètres, le nombre de fichiers est de 256 avec des fichiers dont la taille varie entre 200 Mo et 8Go. Ce relativement faible nombre de fichiers répond à deux problèmes différents. Le premier est le nombre de requêtes nécessaires pour pouvoir faire de la lecture intensive de ces fichiers. Soulignons que plus le nombre de fichiers est élevé, plus le nombre de lectures doit être important. Le second provient de certaines stratégies de réplication de données, qui ont un temps de calcul dépendant du nombre de fichiers et de nœuds pour prendre la décision de réplication. Ces temps de calculs peuvent prendre plusieurs heures en considérant les paramètres proposés.

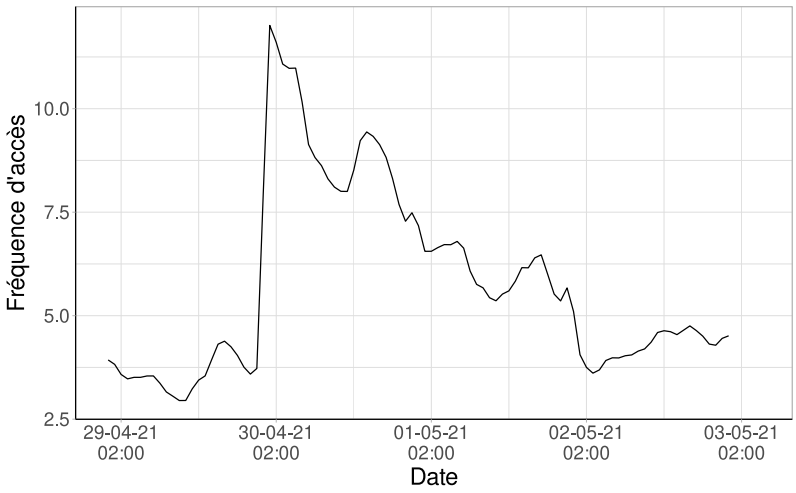


FIGURE 4.8 – Charge de travail de la 2ème expérience (XP2)
Tranche de 1 heure

Comme le montre les documents techniques de Wikipédia [8], l'accès aux articles suit un loi de Zipf entre les 5ème et 1000ème articles les plus lus. Sur cette base, la donnée associée à chaque requête dépend donc d'une loi de Zipf, avec comme paramètre $N = 256$ correspondant au nombre de fichiers et $s = 1$. De plus, l'attribution des tâches au sein de chaque centre se fait à l'aide d'un *round-robin*. C'est-à-dire, que l'ensemble des tâches seront équitablement réparties sur tous les centres de données, mais l'arrivée de ces requêtes est aléatoire comme présenté ci-dessus.

Paramètres	Expérience 1	Expérience 2
Nombre de fichiers	256	256
Taille des fichiers (Go)	[0.2, 4, 8]	[0.2, 4, 8]
Nombre de tâches	150k	2 Millions
Taille des tâches (instructions)	[500, 7.5k]	[500, 7.5k]
Durée des expériences	4h	4 jours
SLO de temps de réponse	15s	15s

TABLE 4.6 – Paramètres d'expérience

4.5 Conclusion

Dans ce chapitre, nous avons introduit l'architecture considérée dans la stratégie de réplication et avons validé les modèles de consommation énergétique présentés dans le chapitre précédent. Nous avons ainsi vu que certains permettaient d'avoir une bonne représentation de la réalité en termes de consommation énergétique. Cependant, nous considérons que ces paramètres permettent de valider notre contribution tout en pouvant la comparer avec d'autres stratégies de la littérature. Les différentes étapes de la stratégie de réplication s'appuient sur cette architecture et sur les modèles introduits précédemment.

Enfin, ces informations sont aussi utilisées dans l'environnement expérimental au travers du simulateur, et de ses paramètres d'entrées. Dans l'environnement expérimental, nous avons aussi présenté les paramètres économiques et énergétiques, ainsi que les charges de travail qui s'appliqueront sur l'architecture, permettant d'évaluer les stratégies de réplication de données mises en place [9].

Dans les deux chapitres suivants, nous présenterons la stratégie de réplication de données proposée. Dans ce qui suit, nous commencerons par la présentation du placement initial sur lequel s'appuie notre stratégie de réplication de données.

8. https ://en.wikipedia.org/wiki/Wikipedia :Does_Wikipedia_traffic_obey_Zipf%27s_law%3F
9. Placement initial – 5.4 et Réplication dynamique – 6.5

Chapitre 5

Placement initial

Sommaire

5.1 Introduction

Dans ce chapitre, nous nous intéressons au placement initial de la stratégie de réplication de données. Pour cela, nous proposons un algorithme permettant de créer et placer des répliques avant l'arrivée des requêtes utilisateurs. Ce placement a pour objectif l'amélioration de la disponibilité et de la tolérance aux fautes pour répondre aux SLOs. Il peut aussi avoir un impact sur les performances ainsi qu'une réduction de la consommation énergétique en plus d'une réduction des dépenses. Ce placement initial, correspond à la réplication statique, sur laquelle viendra s'ajouter la réplication dynamique de la stratégie présentée dans le chapitre 6. Tout au long de ce chapitre, ce placement initial est nommé "*Energy and Expenditure Aware Replication Strategy*" ou *E2ARS*.

Afin de satisfaire ces différents objectifs, nous proposons un algorithme d'optimisation multi-objectifs présenté dans la partie 5.3. Cet algorithme s'appuie sur des fonctions objectifs, qui se basent sur les modèles présentés dans la partie 5.2. Enfin, dans la partie 5.4, nous comparons les différentes possibilités qu'offre ce placement initial et évaluons les performances d'un tel placement en les comparant aux performances d'autres stratégies de réplication de la littérature.

5.2 Formalisation du problème

L'algorithme d'optimisation qu'on présente dans la partie 5.3 se base sur des fonctions objectifs à minimiser. Ces fonctions correspondent à la consommation énergétique et à la dépense. Elles s'appuient sur les modèles présentés dans le chapitre 4.

Dans un premier temps, nous introduisons les notations associées à la formalisation du problème. Puis, nous présentons les modèles utilisés avec les fonctions objectifs ainsi que les contraintes associées.

5.2.1 Notations

Nous reprenons ici quelques notations présentées dans le chapitre 3.

Soit F un ensemble de données notées $f_i \in F$, $1 \leq i \leq z$ et N, un ensemble de nœuds notés $n_j \in N$, $1 \leq j \leq m$. On note cp^j_{disk} la capacité de stockage du nœud.

Soit $\phi(f_i, n_j)$ une matrice de taille (z, m) qui représente le placement des données sur les nœuds.

La variable rt correspond au temps de location des utilisateurs et utilisatrices et $freqRead$ est la fréquence de lecture des usagers durant ce temps de location. Ces paramètres seront des valeurs que les administrateurs et administratrices mettront en entrée de l'algorithme d'optimisation.

Dans les parties suivantes, nous allons présenter les modèles sur lesquels s'appuie la fonction objectif. Sachant que la fonction objectif prend en compte la consommation énergétique et la dépense. Les parties suivantes introduirons dans un premier temps la modélisation des coûts de la réplication, puis du stockage des données et enfin de la lecture des données.

5.2.2 Modélisation de la réplication

Dans un premier temps, nous allons modéliser le coût énergétique et économique de création d'une réplique. Pour cela, nous considérons que chaque usager place ses données sur un nœud, qui est considéré comme le nœud d'origine. L'ensemble des nœuds qui stockent les données d'origine placées par l'utilisateur ou l'utilisatrice est noté $N_o \subset N$. Chaque nœud d'origine, dépendant de la donnée f_i qu'il stocke, est noté n_{i_o}.

Cette prise en compte a un impact sur la modélisation de la réplication. Si le nœud cible correspond au nœud d'origine de la donnée à répliquer, alors la consommation énergétique et la dépense associées à cette réplication sont égales à 0.

Énergie La consommation énergétique associée à la réplication correspond à la consommation énergétique de l'écriture d'une donnée, présentée dans le modèle 3.15, entre le nœud d'origine et le nœud cible de la réplication. L'estimation de la consommation énergétique de l'ensemble des réplications, notée $EC_{\text{replic}}(F, N, N_o, \phi)$, est la suivante :

$$EC_{\text{replic}}(F, N, N_o, \phi) = \sum_{i=1}^{z} \sum_{j=1}^{m} \phi(f_i, n_j) * EC_{write}(n_{i_o}, n_j, f_i) \tag{5.1}$$

Dépense Pour la dépense, nous nous appuyons sur le modèle présenté dans l'équation 3.17. La dépense de l'ensemble des réplications, notée $Cost_{\text{replic}}(F, N, N_o, \phi)$, est estimée de la manière suivante :

$$Cost_{\text{replic}}(F, N, N_o, \phi) = \sum_{i=1}^{z} \sum_{j=1}^{m} \phi(f_i, n_j) * EX_{\text{Network}}(n_{i_o}, n_j, f_i) \tag{5.2}$$

5.2.3 Modélisation du stockage des répliques

Suite à la modélisation de la création de répliques, nous nous intéressons à la modélisation du stockage des répliques.

Énergie Nous nous appuyons sur l'utilisation du *Power Sleep* [Meisner et al., 2009, Wang et al., 2011], qui permet de mettre dans des états de veille les nœuds afin qu'ils consomment moins d'énergie. Nous faisons l'hypothèse qu'un nœud qui stocke des répliques ne peut pas être mis en veille afin de pouvoir répondre aux requêtes le plus rapidement possible. Il est possible que l'ajout d'une réplique puisse légèrement augmenter la consommation énergétique d'un nœud, mais nous ne considérons pas cette augmentation dans nos modèles.

Cela implique que stocker une réplique sur un nœud vide, consomme plus que de la stocker sur un nœud stockant déjà des répliques. Un nœud qui ne stocke pas de répliques peut alors être éteint, dans notre modèle cette consommation est égale à 0. Le modèle de consommation énergétique proposé se base sur l'équation 3.7 et se nomme $EC_{\text{store}}(F, N, rt, \phi)$. Ce modèle est le suivant :

$$EC_{\text{store}}(F, N, rt, \phi) = \sum_{j=1}^{m} EC_{static}(\emptyset, n_j, rt) * \\ [1 - \prod_{i=1}^{z}(1 - \phi(f_i, n_j))] \tag{5.3}$$

Dépense Du point de vue économique, le coût de stockage d'une réplique, dépend du nœud, et de la quantité de répliques stockées sur le nœud. Le modèle $Cost_{\text{store}}(F, N, rt, \phi)$, se basant sur l'équation 3.16, est calculé de la manière suivante :

$$Cost_{\text{store}}(F, N, rt, \phi) = \sum_{i=1}^{z}\sum_{j=1}^{m} \phi(f_i, n_j) * EX_{\text{Storage}}(n_j, f_i, rt) \tag{5.4}$$

5.2.4 Modélisation de la lecture de données

Les modèles présentés dans cette partie ont pour but d'estimer la consommation énergétique et la dépense moyennes d'accès à une donnée, par des nœuds qui n'en possèdent pas la réplique.

Soit N_i, un ensemble de nœuds qui stockent les répliques de la donnée f_i et $N_{\bar{i}}$ les nœuds qui ne stockent pas cette donnée. Ces ensembles sont différents pour chaque donnée et sont construits sur $\phi(f_i, n_j)$. En effet, N_i représente les nœuds où $\phi(f_i, n_j) = 1$ pour une donnée f_i et $N_{\bar{i}}$ contient les autres nœuds.

Dans cette partie, chaque nœud de l'ensemble N_i est noté n_j avec $1 \leq j \leq m_i$ et m_i est égal au nombre de répliques de f_i. Pour $N_{\bar{i}}$, les nœuds sont notés $n_{j'}$ avec $1 \leq j' \leq m_{\bar{i}}$.

Pour avoir une idée de l'impact de chaque lecture sur la consommation énergétique et la dépense, il est possible de tenir compte uniquement de la moyenne, de la médiane, ou encore du maximum pour chaque lecture. La moyenne a l'avantage de pouvoir représenter un coût de lecture pour la prise de décision par les administrateurs et administratrices, selon les valeurs $freqRead$ et rt mises en entrée.

Énergie En s'appuyant sur l'équation 3.14, le modèle $EC_{\text{AvgRead}}(F, N, \phi)$, correspondant à la consommation moyenne, est calculé de la manière suivante :

$$EC_{\text{AvgRead}}(F, N, \phi) = \frac{1}{z}\sum_{i=1}^{z}[\frac{1}{m_{\bar{i}}}\sum_{j'=1}^{m_{\bar{i}}} * EC_{Read}(N_i, n_{j'}, f_i)] \tag{5.5}$$

Dépense Comme pour le modèle de coût d'écriture présenté plus haut, nous considérons comme coûts de lecture, les coût de transferts. Pour cette estimation de la moyenne, la fonction $short(n_j, n_{j'}, f_i)$, introduite pour l'équation 3.14, retourne le nœud n_j qui stocke f_i le plus proche de $n_{j'}$ en terme de bande-passante. Le modèle $Cost_{\text{AvgRead}}(F, N, \phi)$, estimant la moyenne des coûts, s'appuie sur l'équation 3.17 :

$$Cost_{\text{AvgRead}}(F, N, \phi) = \frac{1}{z} \sum_{i=1}^{z} [\sum_{j=1}^{m_i} \frac{1}{m_{\bar{i}}} \sum_{j'=1}^{m_{\bar{i}}} short(n_j, n_{j'}, f_i) * s(f_i) * Pr_{network}(n_j, n'_j)] \tag{5.6}$$

5.2.5 Fonctions Objectifs

Les modèles précédemment introduits sont regroupés dans des fonctions objectifs de consommation énergétique et de dépense. Ces fonctions permettent de formaliser le problème auquel devra répondre l'algorithme d'optimisation. Comme cela a été introduit précédemment, les paramètres rt et $freqRead$ correspondent respectivement au temps de location et à la fréquence de lecture que la ou le locataire fera durant ce temps de location. Ces paramètres sont des hypothèses que les administrateurs et administratrices prennent en compte pour ajuster le nombre de répliques. Plus le temps de location est élevé avec une fréquence de lecture faible, moins il y aura de répliques pour réduire, sur le long terme, la consommation énergétique et la dépense. L'impact de ces paramètres sera analysé par la suite dans la partie 5.4.1.

Énergie En se basant sur les modèles présentés dans les équations (5.1),(5.3) et (5.5), la fonction objectif de consommation énergétique est la suivante :

$$ECG(F, N, N_0, rt, freqRead, \phi) = EC_{replic}(F, N, N_o, \phi) + EC_{storage}(F, N, rt, \phi) + rt * freqRead * EC_{read}(F, N, \phi) \tag{5.7}$$

Dépense De même, en se basant sur les équations (5.2), (5.4) et (5.6), la fonction objectif de dépense utilisée est :

$$ExpG(F, N, N_0, rt, freqRead, \phi) = Cost_{replic}(F, N, N_o, \phi) + Cost_{storage}(F, N, rt, \phi) + rt * freqRead * Cost_{read}(F, N, \phi) \tag{5.8}$$

Formalisation du problème Pour résumer l'ensemble du problème, la réplication vise à minimiser les fonctions objectifs de consommation énergétique et de dépense. Nous avons comme entrées l'ensemble des nœuds de notre infrastructure (N), l'ensemble des données à répliquer (F) ainsi que leurs nœuds d'origines (N_o). Les administrateurs et administratrices peuvent modifier les paramètres rt et $freqRead$, correspondant aux hypothèses de temps de location et fréquence de lecture respectivement. Enfin, la variable de décision ϕ est une matrice représentant le placement des répliques de chaque donnée f_i sur les nœuds n_j.

Dans notre problème, nous considérons quelques contraintes importantes. (i) La taille totale des répliques à stocker sur un nœud ne peut pas dépasser sa capacité de stockage cp^j_{disk}. (ii) Un nombre minimum de 2 répliques doivent être créées pour chaque donnée.

Cette seconde contrainte provient de [Li et al., 2011], qui estiment que 3 répliques assurent une disponibilité de plus de 99.99%, si les nœuds qui les stockent sont assez indépendants, dans une année par rapport à la probabilité de panne d'un nœud.

La formalisation du problème s'appuie sur les fonctions objectifs (5.7) et (5.8), et est présentée de la manière suivante :

$$\begin{cases} \min\limits_{\phi} & ECG(F, N, N_0, rt, freqRead, \phi) \\ \min\limits_{\phi} & ExpG(F, N, N_0, rt, freqRead, \phi) \\ s.c. : & \sum\limits_{i=1}^{z} \phi(f_i, n_j) * s(f_i) \leq cp^j_{disk} \quad ,\forall j \\ & \sum\limits_{j=1}^{m} \phi(f_i, n_j) \geq 3 \quad ,\forall i \end{cases} \tag{5.9}$$

Ces contraintes et fonctions objectifs permettent de valider chaque solution et de les évaluer du point de vue de la consommation énergétique et de la dépense. Nous allons à présent introduire le fonctionnement du placement initial des données.

5.3 Algorithme d'optimisation

Dans cette partie, nous allons présenter l'algorithme de placement initial. Nous commencerons ainsi par une présentation globale de cet algorithme, qui est composé d'un placement inter-centres de données suivi d'un placement intra-centre de données. Chacune de ces parties est ensuite décrite dans les sous-parties 6.3.2 et 6.3.3 respectivement.

5.3.1 Introduction

Dans cette partie, nous allons présenter le fonctionnement de l'algorithme de réplication initial. Nous utilisons un algorithme d'optimisation multi-objectif. Cependant, dans un contexte large échelle avec des milliers de nœuds, considérer tous les nœuds dans l'espace de recherche, pourrait augmenter de manière significative le temps de calcul de l'algorithme. Pour répondre à ce problème de temps de réponse, la procédure de décision de réplication se fait en 2 étapes décrites dans l'algorithme 1.

Les variables utilisées dans cet algorithme sont présentées sur le tableau 5.1. Certaines variables utilisées sont des objets qui contiennent plusieurs informations. Dans les objets se trouvent les **nœuds**, qui ont un identifiant unique (idN), une capacité de stockage (cp), un identifiant du centre auquel ils appartiennent (DCid) et enfin un ensemble d'informations (infos) de composants, de consommation énergétique et de dépense, nécessaires pour les algorithmes d'optimisation. De même, une **donnée** est un autre objet, qui possède un identifiant unique (idF), une taille (size) et un nœud de stockage d'origine. Un résumé de ces objets est présenté dans le tableau 5.2.

Cet algorithme commence par initialiser la matrice ϕ, en la remplissant de zéros (ligne 1), et récupère ensuite les nœuds sur lesquels sont stockés les données avant la réplication (ligne 2).

La première étape consiste à réduire l'espace de recherche en considérant uniquement les centres de données. Cette étape décide des centres de données sur lesquels chaque donnée sera répliquée. Pour cela, elle sélectionne un ou plusieurs représentants de chaque centre de données (ligne 3). Nous supposons qu'au sein d'un centre de données, les nœuds sont homogènes, et que l'hétérogénéité s'applique entre les centres de données. Ainsi, nous prenons un seul représentant par centre de données. Le choix des représentants correspond à la fonction getRepresentativesDC(N) (ligne 3) de l'algorithme 1 et l'algorithme d'optimisation correspond à la fonction getSPEA2Result(F, DCrep, OriginFiles, rt, freqRead) (ligne 4) présenté dans la partie 5.3.2. Cet algorithme retourne une liste de solutions issues du front de Pareto. Les administratrices et administrateurs décident de la solution à appliquer avec la fonction chooseIndividual(listInds) (ligne 5).

La seconde étape décide, au sein de chaque centre de données, sur quels nœuds seront stockées les répliques issues de la première étape. Pour cela, l'algorithme récupère les nœuds et les données à placer de chaque centre (lignes 7-8). Puis, il place les données à répliquer sur les nœuds, correspondant à la fonction **FirstFitNoBottleneck**(F, F_{DC}, N_{DC}) (ligne 9). Cette fonction retourne une matrice de placement des données à répliquer sur les nœuds du centre de données. La dernière fonction remplace dans la matrice ϕ le résultat de cette fonction, pour les nœuds du centre de données. Cette étape est appliquée à chaque centre de données.

Variable	Type	Description
ϕ	Matrice d'entiers	Matrice de placement des données sur les nœuds
chInd	Matrice d'entiers	Matrice solution de la réplication inter-centres
DC	Entier	Identifiant d'un centre de données
DcRep	Liste de **Nœuds**	Liste des nœuds représentant les centres de données
F	Liste de **Données**	Liste de données à répliquer
F_{DC}	Liste de **Données**	Liste des données à stocker dans le centre
freqRead	Flottant	Hypothèse de fréquence de lecture
Ind_{DC}	Matrice d'entier	Matrice de placement des données sur les nœuds pour un centre
listInds	Liste de matrices d'entiers	Liste de matrices représentant les solutions proposées par SPEA2, ces matrices correspondent à la réplication entre centres de données
listOfDC	Liste d'entiers	Liste d'identifiants des centres de l'architecture
N	Liste de **Nœuds**	Liste des nœuds de l'architecture
N_{DC}	Liste de **Nœuds**	Liste des nœuds d'un centre
OriginFiles	Dictionnaire d'entier :**Nœud**	Dictionnaire qui prend comme clé le numéro de la donnée et comme valeur le nœud qui la stocke avant la réplication
rt	Entier	Hypothèse de temps de location

TABLE 5.1 – Description des variables utilisées dans l'algorithme 1

Objet	Variable : Type	Description
Donnée	idF : Entier	Identifiant de la donnée
	size : Entier	Taille de la donnée
	origin : Entier	Identifiant du nœud qui stocke la donnée avant la réplication
Nœud	idN : Entier	Identifiant du nœud
	cp : Entier	Capacité de stockage
	DCid : Entier	Identifiant du centre dans lequel il est
	infos : Texte	Différentes informations associées au nœud

TABLE 5.2 – Description des objets de l'algorithme 1

5.3.2 Réplication de données inter-centres de données

Cette première étape de décision est importante, car la réplication se fait entre différents centres de données ayant des caractéristiques différentes. Elle correspond aux lignes 1-5 de l'algorithme 1. L'algorithme doit donc répliquer et placer ces répliques afin d'équilibrer au mieux le compromis entre la consommation énergétique et la dépense, du point de vue des administrateurs et administratrices.

Nous supposons dans cette première étape, qu'un centre de données a la capacité de stocker toutes les données de F. La contrainte de capacité n'est, de ce fait, pas appliquée ici. Cependant, nous appliquons la contrainte du nombre minimum de 3 répliques. En effet, la stratégie de réplication de données proposée par [Li et al., 2011] essaie de retarder au maximum la réplication pour réduire les dépenses, tout en considérant les problématiques de disponibilité. Cette stratégie met en avant l'importance de répliquer les données sur des nœuds qui sont les plus indépendants possibles afin de réduire la probabilité de perdre une donnée à cause de la panne d'un nœud. Par exemple, si 2 répliques sont stockées dans le même centre de données et qu'il y a une coupure de courant dans ce centre de données, alors ces répliques deviennent inaccessibles.

Pour considérer ces deux objectifs, il est possible de considérer un objectif comme une contrainte et de minimiser l'autre objectif [Laumanns et al., 2006], ou de faire une fonction objectif unique qui applique des poids à chaque objectif [Long et al., 2014]. Cependant, nous avons choisi d'utiliser des algorithmes qui cherchent le front de Pareto. Cela permet de laisser aux administrateurs et administratrices la décision de s'il est plus important de réduire la consommation énergétique, la

Procedure 1 Algorithme de placement initial

Input: N : Liste de nœuds
Input: F : Liste de données
Input: rt : temps de location
Input: freqRead : fréquence de lecture
Input: listOfDC : Liste de centres de données DC
Output: ϕ : matrice de données et nœuds qui représente le placement des répliques

1: ϕ = zeros(length(F),length(N))
{matrice (nombre de données, nombre de nœuds) de valeurs nulles}
2: OriginFiles = getOriginalNodeFile(F, N)
{Nœuds qui stockent les données avant l'algorithme}
3: DcRep = getRepresentativesDC(N)
{Selection d'un nœud pour chaque centre de données}
4: listInds = getSPEA2Result(F, DCrep, OriginFiles, rt, freqRead)
5: Ind = chooseIndividual(listInds)
6: **for** DC in listOfDC **do**
7: N_{DC} = getNodesFromDC(N, DC)
8: F_{DC} = getFileOnDC(Ind, F)
9: Ind_{DC} = **FirstFitNoBottleneck**(F, F_{DC}, N_{DC})
10: ϕ = replace(ϕ, Ind_{DC})
{fonction qui remplace les valeurs de ϕ pour les nœuds du centre de données, par le résultat de FirstFitNoBottleneck}
11: **end for**

dépense, ou de prendre une décision plus équilibrée. Cette étape de décision correspond à la fonction chooseIndividual(DC_individuals) dans la ligne 5. Ce choix peut être appuyé par les valeurs de consommation énergétique et de dépense, qui permet de voir les différences entre les possibilités données par l'algorithme.

Pour cela, nous utilisons l'algorithme d'optimisation évolutionnaire, *Improved Strength Pareto Evolutionary Algorithm* ou SPEA2 [Zitzler et al., 2001] (correspondant à getSPEA2Result de la ligne 4 dans l'algorithme 1). Les algorithmes évolutionnaires fonctionnent sur la base d'une population d'"individus" qui ont des gènes leur étant propres. Les "individus" dans notre cas, sont différentes possibilités de ϕ et un gène correspond à une valeur de cette matrice. SPEA2 divise ces individus en deux catégories. Les "individus" dominés, correspondent à des individus pour lesquels il existe un individu qui a au moins les mêmes résultats sur tous les objectifs, et a de meilleurs résultats sur au moins un objectif. Et les "individus" qui dominent, c'est-à-dire qu'ils ne sont dominés par aucun autre individu. Les "individus" qui dominent sont placés dans une archive qui répertorie les meilleurs individus du point de vue des objectifs.

L'algorithme va enregistrer les individus les plus adaptés dans une archive, dont la taille est définie par l'administrateur ou l'administratrice. Si le nombre d'individus est supérieur à la taille de l'archive, alors SPEA2 va calculer la distance entre les dominants et retirer les individus qui sont trop proches entre eux. L'algorithme SPEA2 fonctionne en 6 étapes :

1. Création d'une population d'individus aléatoire
2. Calcul du degré d'adaptation pour chaque individu
3. Sélection des individus en se basant sur la dominance et en gardant une partie des meilleurs individus dominés (selon la place dans l'archive)
4. Fin de l'algorithme si le nombre maximum de générations est atteint
5. Tirage aléatoire avec remise dans l'archive pour créer un ensemble de reproduction
6. Opérations de recombinaison et de mutation dans l'ensemble de reproduction, puis on repart de l'étape 2

Dans le cadre des algorithmes évolutionnaires, la partie mutation et recombinaison est particulièrement importante car elle permet de créer la diversité des individus, donc des résultats qui permettent d'approcher petit à petit le front de Pareto. Il existe tout un champ de recherche qui a pour objectif d'améliorer les techniques de mutation et de recombinaison [Aguirre and Tanaka, 2005, Sato et al., 2005, Sato et al., 2011].

Nous appliquons deux techniques différentes dans nos opérations de mutations et de recombinaison : la mutation et le contrôle des gènes croisés pour un croisement uniforme (CCG_{UX}) [Sato et al., 2011].

Mutations Comme le tirage de l'ensemble de reproduction est aléatoire avec remise, il est possible que 2 individus parents soient les mêmes. Si c'est le cas, alors on applique une mutation sur chaque parent indépendamment. C'est-à-dire que sur chaque valeur de la matrice ϕ, il y a une probabilité de mutation qui est mise en place. Lorsque la mutation arrive sur la case de la donnée f_i et du nœud n_j, si $\phi(f_i, n_j) = 0$ alors la mutation donnera $\phi(f_i, n_j) = 1$, et vice versa. Cela permet de générer 2 nouveaux individus provenant de la mutation des parents.

Recombinaison (CCG_{UX}) Si les 2 individus parents sont différents, alors on applique le CCG_{UX}. L'idée de cette technique est : pour chaque case de ϕ, il y a une probabilité d'échange entre les deux parents. Ainsi, soient ϕ et ϕ' les parents 1 et 2 respectivement, lorsque l'échange arrive pour la donnée f_i et le nœud n_j alors, $\phi(f_i, n_j)$ prend la valeur de $\phi'(f_i, n_j)$ et vice versa.

Dans notre cas, nous avons appliqué cette technique sur l'ensemble des nœuds qui stockent la donnée, afin d'assurer la contrainte de disponibilité. Chaque donnée a une probabilité de recombinaison, et lorsque la recombinaison arrive sur une donnée f_i, alors les nœuds qui stockent les répliques s'échangent entre les parents 1 et 2.

L'algorithme 2, permet de résumer les techniques de mutation (lignes 7-27) et de recombinaison (lignes 29-36) utilisées dans l'application de SPEA2. Pour en faciliter la lecture, nous proposons le tableau 5.3 de description des variables.

Variable	Type	Description
ϕ, ϕ'	Matrice d'entiers	Matrices de placement des données sur les nœuds, considérées comme des individus
F	Liste de **Données**	Liste de données
ID	Entier	Indice dans la liste de reproduction
matingPool	Liste de Matrices d'entiers	Liste d'individus (ϕ) de reproduction
N	Liste de **Nœuds**	Liste des nœuds de l'architecture
offspringPop	Liste de Matrices d'entiers	Liste d'individus (ϕ) issus de la reproduction
poolSize	Entier	Taille de la liste de reproduction
probRecomb	Flottant	Probabilité donnée à la recombinaison
probVariation	Flottant	Probabilité donnée à la mutation d'un gène
random	Librairie pour l'aléatoire	Librairie qui permet de générer des valeurs aléatoires
rng	Flottant	Valeurs aléatoire entre 0 et 1
saveFile	Liste d'entier	Liste de Sauvegarde de nœuds qui stockent une donnée

TABLE 5.3 – Description des variables utilisées dans l'algorithme 2

À la fin de SPEA2, la stratégie retourne l'ensemble des individus de l'archive et les administratrices et administrateurs décideront la politique à mettre en place. Ce choix se fera parmi un spectre de possibilités provenant du front de Pareto, qui va du fait de tenir compte uniquement des dépenses jusqu'à tenir compte uniquement de la consommation énergétique. Cette décision est représentée avec la fonction chooseIndividual(DC_individuals) (ligne 5) de l'algorithme 1.

Après avoir décidé quels centres de données stockera quelles répliques, l'algorithme va ensuite décider sur quels nœuds seront stockées les répliques dans les centres de données choisis.

Procedure 2 Techniques de mutation et recombinaison

Input: N : Liste de nœuds
Input: F : Liste de données
Input: matingPool : Liste d'individus de reproduction
Output: offspringPop : Liste d'individus issues de la reproduction

probVariation = getVariationProbability()
probRecomb = getRecombinationProbability()
poolSize = length(matingPool)
for ID in 0 :poolSize/2 **do**
  ϕ = matingPool[ID*2]
  ϕ' = matingPool[ID*2 + 1]
  if $\phi == \phi'$ **then**
    repeat
      for f_i in F **do**
        for n_j in N **do**
          rng = random.uniforme(0,1)
          if rng < probVariation **then**
            $\phi(f_i, n_j)$ = MODULO($\phi(f_i, n_j)$+1, 2)
          end if
        end for
      end for
    until checkConstraint(ϕ)
    {Vérification du nombre minimum de répliques pour ϕ}
    repeat
      for f_i in F **do**
        for n_j in N **do**
          rng = random.uniforme(0,1)
          if rng < probVariation **then**
            $\phi'(f_i, n_j)$ = MODULO($\phi'(f_i, n_j)$+1, 2)
          end if
        end for
      end for
    until checkConstraint(ϕ')
    {Vérification du nombre minimum de répliques pour ϕ'}
  else
    for f_i in F **do**
      rng = random.uniforme(0, 1)
      if rng < probRecomb **then**
        saveFile = $\phi(f_i, :)$ {Sauvegarde des nœuds qui stockent f_i}
        $\phi(f_i, :) = \phi'(f_i, :)$
        $\phi'(f_i, :)$ = saveFile
      end if
    end for
  end if
end for

5.3.3 Réplication de données intra-centre de données

Cette seconde étape a pour but de choisir quels nœuds stockeront les répliques que devra stocker le centre de données. Elle correspond aux lignes 6-11 de l'algorithme 1 et est appliquée à chaque centre de données. Avant d'appliquer l'algorithme de placement, nous devons récupérer les nœuds du centre de données (correspondant à la fonction getNodesFromDC (ligne 7) de l'algorithme 1) ainsi que les répliques qui seront stockées dans ce centre de données (correspondant à la fonction getFileOnDC (ligne 8) dans l'algorithme 1).

En s'appuyant sur les états de veille, nous avons décidé de placer les données sur un petit nombre de nœuds afin de pouvoir mettre en veille les autres nœuds permettant ainsi de réduire la consommation énergétique. La différence de coût s'appliquant principalement entre les centres de données, placer les données sur n'importe quel nœud ne changerait pas les coûts (comme les nœuds sont homogènes dans un centre de données).

Nous considérons ainsi une proportion de nœuds qui stockeront ces répliques, qui formeront un sous-ensemble de nœuds de stockage. Ainsi, si la taille totale des répliques à stocker dans le centre de données, est supérieure à la capacité de stockage de ce sous-ensemble de nœuds, alors on ajoute la même proportion de nœuds à ce sous-ensemble, jusqu'à avoir un nombre de nœuds suffisant. Cette méthode n'est probablement pas optimale, mais la solution est suffisante au vu du temps de calcul.

Cependant, le choix de cette proportion a un impact important à la fois sur la consommation énergétique et sur les performances. Comme nous le verrons dans l'évaluation 5.4.3, plus ce pourcentage est faible, plus cela crée un goulot d'étranglement et génère beaucoup de violations. À l'inverse, si la proportion est trop élevée, alors une grande partie des nœuds devront rester allumés, ce qui augmente grandement la consommation énergétique.

Lorsque le nombre de nœuds est défini, les répliques sont triées de manière décroissante selon leurs tailles, et sont placées sur les nœuds en utilisant l'algorithme *round-robin*.

Variable	Type	Description
ϕ_{DC}	Matrice d'entiers	Matrice de placement des données sur les nœuds du centre de données
capacity	Liste d'entiers	Liste de capacité de stockage des nœuds du centre de données
F	Liste de **Données**	Liste de données
F_{DC}	Liste de **Données**	Liste de données à stocker dans le centre de données
fileStored	Booléen	Booléen permettant de vérifier que la donnée a bien été stockée
idNode	Entier	Identifiant du nœud sur lequel sera placée la donnée en premier
N_{DC}	Liste de **Nœuds**	Liste des nœuds du centre de données
nbFile	Entier	Compteur de nombre de données placées
nbNodeTested	Entier	Nombre de nœuds sur lesquels le placement aura été testé
sortedFileIds	Liste d'entiers	Liste des identifiants de données triées par taille (du plus léger au plus lourd)
totalNbNode	Entier	Nombre de nœuds du centre qui stockeront les données

TABLE 5.4 – Description des variables utilisées dans l'algorithme 3

L'algorithme FirstFitNoBottleneck qui réalise ce placement, est décrit dans l'algorithme 3. Pour en faciliter la lecture nous proposons le tableau 5.4 de description des variables. Cet algorithme prend en entrée la liste des données, les répliques qui seront stockés dans le centre de données ainsi que la liste de nœuds du centre de données. Dans un premier temps, l'algorithme va trier les répliques selon leurs tailles (ligne 2). Puis, à l'aide de la taille des répliques, et de la proportion de nœuds de stockage, il récupère le nombre de nœuds qui permettent de stocker l'ensemble des données (correspondant à la fonction getTotalNbNode à la ligne 3). L'algorithme va ensuite récupérer les

Procedure 3 FirstFitNoBottleneck

Input: F : Liste des données
Input: F_{DC} : Liste des données qui seront répliquées dans le centre de donnée
Input: N_{DC} : Liste des nœuds du centre de données
Output: ϕ_{DC} : Matrice de données et nœuds qui représente le placement des répliques dans un centre

```
1: φ_DC = zeros(length(F), length(N_DC))
   {matrice (nombre de données, nombre de nœuds) de valeurs nulles}
2: sortedFileIds = sort(F_DC)
   {Trie des données en se basant sur leurs tailles et retourne leurs identifiants}
3: totalNbNode = getTotalNbNode(N_DC, F_DC)
4: capacity = getAllCapacity(N_DC)
   {Retourne une liste de capacité des nœuds du centre de données}
5: nbFile = 0
6: for idFile in sortedFilesListId do
7:    idNode = MODULO(nbFile, totalNbNode)
      {Premier nœud choisi}
8:    tempNode = N_DC[idNode]
9:    fileStored = false
10:   nbNodeTested = 0
11:   repeat
12:      nbNodeTested += 1
13:      if capacity[tempNode] ≥ s(F[idFile]) then
14:         Ind_DC[idFile][idNode] = 1
15:         fileStored = true
16:         capacity[tempNode] -= s(F[idFile])
17:      else
18:         idNode = MODULO(idFile + nbNodeTested, totalNbNode)
19:         tempNode = N_DC[idNode]
20:      end if
21:   until fileStored ∨ nbNodeTested ≥ totalNbNode
22:   if ¬ fileStored then
23:      idNode = totalNbNode
24:      while ¬ fileStored do
25:         tempNode = N_DC[idNode]
26:         if capacity(tempNode) ≥ s(F[idFile]) then
27:            φ_DC[idFile][idNode] = 1
28:            fileStored = true
29:         end if
30:         idNode += 1
31:      end while
32:   end if
33:   nbFile += 1
34: end for
```

capacités de stockages des nœuds sélectionnés (ligne 4). Chaque réplique sera placée à l'aide d'un *round-robin* (lignes 11-21). Cependant, si la réplique ne peut être stockée par manque de capacité sur un nœud, l'algorithme va la placer sur le nœud suivant (lignes 22-32).

Nous allons à présent vérifier l'efficacité de ce placement initial à travers des expériences. Nous analysons par la suite l'impact des différentes politiques possibles.

5.4 Évaluation des performances

Dans cette partie, nous allons présenter différents résultats relatifs au placement initial de la stratégie de réplication de données proposée. Pour cela, nous nous appuyons sur les expériences courts et longs termes présentées dans la partie 4.4 du chapitre 4.

Le premier résultat présenté concernera l'impact de différents choix de paramètres et de politiques du placement initial. Puis, nous analyserons l'impact du choix de la proportion de nœuds à garder allumés correspondant à l'hypothèse présentée dans la partie 5.3.3. Enfin, nous comparerons ce placement avec d'autres stratégies de réplication de données issues de la littérature.

5.4.1 Evaluation des paramètres *rt* et *freqRead*

Dans un premier temps, nous allons analyser l'impact du choix des paramètres du placement entre les centres de données, décrit dans la partie 5.3.2. Ces paramètres permettent de choisir si (i) la réplication se fera dans un contexte avec une faible charge de travail sur du long terme (*Energy and Expenditure Aware Replication Strategy – Long Term* ou *E2ARSLT*), ou (ii) avec une forte charge de travail sur du court terme (*Energy and Expenditure Aware Replication Strategy – Short Term* ou *E2ARSST*). Enfin, nous comparons ces choix de paramètres avec (iii) un choix intermédiaire : *E2ARS*.

Les paramètres de (i) *E2ARSLT* correspondent à une location de 10 mois pour 250k lectures. À l'inverse, les paramètres de (ii) *E2ARSST* correspondent à une location de 3 jours pour 1 million de lectures. Enfin, les paramètres intermédiaires de (iii) *E2ARS* tiennent compte d'une location de 1 mois pour 500k lectures. Un résumé de ces choix est présent dans le tableau 5.5.

Nom	$rt =$	$freqRead =$
E2ARS	30 jours	16.6k lecture/jour
E2ARSST	3 jours	333k lectures/jour
E2ARSLT	300 jours	833 lectures/jour

Table 5.5 – Différences entre les choix de paramètres

Nous avons comparé ces choix de paramètres avec les expériences présentées dans le chapitre 4, représentant une charge normale, avec des expériences moins chargées (*Faible charge*). Ces expériences de faible charge ont la même durée que les expériences originales, mais la charge de travail est divisée par 5. Cela correspond aux paramètres présentés dans le tableau 5.6.

Environnement	Charge normale		Faible charge	
Expérience	1	2	1	2
Tâches	150k	2 millions	30k	400k
Durée	4h	4 jours	4h	4 jours

Table 5.6 – Différences entre les expériences à charge normale et les expériences à faible charge

Sur le tableau 5.7, représentant le nombre de répliques créées, nous voyons que E2ARSLT est le choix qui crée le moins de répliques, avec une réduction de 40% du nombre de répliques par rapport à E2ARS. À l'inverse, E2ARSST crée seulement 4% de répliques supplémentaires par rapport à E2ARS.

Paramètres	Réplications
E2ARS	879.4 (121.8)
E2ARSST	914.9 (106.7)
E2ARSLT	519.7 (7.1)

TABLE 5.7 – Nombre de répliques créées sur toutes les expériences pour chaque choix de paramètres
Moyenne (écart-type)

	Env.	Param.	Violations	Énergie (MJ)	Dépense (k$)
Expérience 1	Faible charge	E2ARS	2.65% (0.76%)	157.1 (0.1)	3 (0.3)
		E2ARSST	2.42% (0.67%)	157.1 (0.1)	3 (0.3)
		E2ARSLT	4.7% (0.22%)	156.8 (0.1)	4 (0.1)
	Charge normale	E2ARS	58.4% (2.5%)	232.5 (38)	14 (1.8)
		E2ARSST	58.3% (2.5%)	221.8 (37)	13 (1.4)
		E2ARSLT	64.2% (1.5%)	334.6 (18)	19 (0.9)
Expérience 2	Faible charge	E2ARS	0.01% (0%)	3751.5 (0.2)	36 (3.4)
		E2ARSST	0% (0%)	3751.5 (0.3)	36 (3.7)
		E2ARSLT	0.01% (0%)	3750.6 (0.1)	47 (1.9)
	Charge normale	E2ARS	16.7% (3.5%)	3769.9 (2.2)	201 (22.4)
		E2ARSST	16.9% (2.6%)	3769.2 (0.9)	199 (14.4)
		E2ARSLT	23.6% (1.5%)	3782.4 (7.6)	255 (6)

TABLE 5.8 – Résultats en terme de pourcentage de violations, de la consommation énergétique et de la dépense par expérience, pour chaque choix de paramètres
Moyenne (écart-type)

Le tableau 5.8 résume les résultats en termes de violations, de consommation énergétique et de dépense, dans chaque expérience, pour chacun des choix de paramètres. Le nombre de violations est lié ici au nombre de répliques créées. En effet, les choix avec le plus de répliques, sont ceux qui ont généré le moins de violations.

Les résultats mettent en avant l'importance des choix de paramètres, principalement en termes de dépense. Dans les environnements expérimentaux à faible charge, sur la première expérience de 4h, nous voyons que la consommation énergétique de E2ARSLT est légèrement plus faible que E2ARS de l'ordre de 0.2% avec une dépense plus élevée de l'ordre de 33%. Nous voyons le même comportement avec l'expérience à long terme, où la réduction de la consommation par rapport à E2ARS est inférieure à 1% avec une augmentation de 31% de la dépense. Le choix de paramètre E2ARSST consomme et coûte autant que E2ARS dans l'environnement à faible charge.

Dans les expériences à charge normale, nous voyons que E2ARSLT n'est pas adapté. Sur toutes les expériences, elle augmente à la fois la consommation énergétique et la dépense. Dans la première expérience, ces augmentations sont à hauteur de 33% et 25% respectivement par rapport à E2ARS. Alors que dans la seconde, la consommation énergétique et la dépense augmentent de 0.3% et 27.4% respectivement par rapport à E2ARS. Dans ces expériences, E2ARSST réduit à la fois la consommation énergétique et la dépense dans les 2 expériences par rapport à E2ARS, même si cette diminution est moins importante que l'augmentation induite par E2ARSLT.

Cette augmentation de la dépense quelque soit l'environnement expérimental s'explique par le fait qu'une grande partie de cette dépense est associée au transfert de données. Comme cela se voit sur la table 5.9, pour l'ensemble des choix, le coût du réseaux correspond à 99% de la dépense totale. Or, E2ARSLT crée peu de répliques, nécessitant plus de transferts de données pour répondre aux requêtes, que E2ARSST qui en crée plus. La réduction de coûts associés au stockage entre E2ARSLT et E2ARSST n'est pas aussi importante que l'augmentation des coûts de transfert.

Paramètres	Réseaux ($)	Stockage ($)	Exécution ($)	Pénalité ($)
E2ARS	3366	3.33	0.3	29.03
E2ARSST	3308	3.48	0.3	26.16
E2ARSLT	3983	1.92	0.3	54.25

TABLE 5.9 – Répartition des coûts par choix de paramètres

Pour la suite, nous garderons l'hypothèse d'une location d'un mois avec 500k lectures, correspondant au choix intermédiaire (E2ARS).

5.4.2 Choix de la politique

Comme nous l'avons décrit dans la partie 5.3.2, l'algorithme d'optimisation SPEA2, fournit un ensemble d'individus issus du front de Pareto. Ces individus représentent différentes politiques possibles à appliquer. Les résultats à la sortie de l'algorithme sont représentés sur la figure 5.1. Sur la base de ces résultats, trois politiques sont comparées entre elles :

— Une politique qui considère uniquement la consommation énergétique (*E2ARSEC*)
— Une politique qui considère uniquement la dépense (*E2ARSEX*)
— Une politique intermédiaire (*E2ARS*)

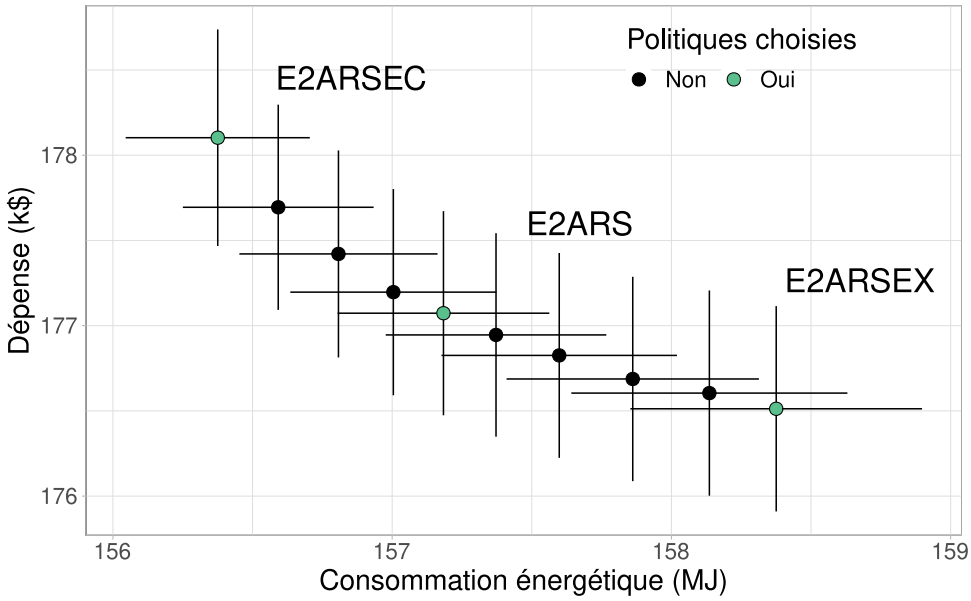


FIGURE 5.1 – Front de Pareto à la sortie de SPEA2

Paramètres	Réplications
E2ARS	879.4 (121.8)
E2ARSEC	863.5 (106.8)
E2ARSEX	901.1 (117.8)

TABLE 5.10 – Nombre de répliques créées sur toutes les expériences pour chaque choix de politique
Moyenne (écart-type)

	Env.	Pol.	Violations	Énergie (MJ)	Dépense (k$)
Expérience 1	Faible charge	E2ARS	2.65% (0.76%)	157.1 (0.1)	3 (0.3)
		E2ARSEC	2.95% (0.59%)	157 (0.1)	4 (0.3)
		E2ARSEX	2.43% (0.69%)	157.1 (0.1)	3 (0.3)
	Origine	E2ARS	58.44% (2.49%)	232.5 (38)	14 (1.8)
		E2ARSEC	58.98% (2.27%)	251.1 (38.6)	15 (1.6)
		E2ARSEX	58.79% (1.99%)	230.1 (35.1)	14 (1.5)
Expérience 2	Faible charge	E2ARS	0.01% (0%)	3751.5 (0.2)	36 (3.4)
		E2ARSEC	0% (0%)	3751.5 (0.2)	36 (4)
		E2ARSEX	0% (0%)	3751.6 (0.3)	33 (4.2)
	Origine	E2ARS	16.69% (3.54%)	3769.9 (2.2)	201 (22.4)
		E2ARSEC	18.59% (2.66%)	3769.6 (2.6)	212 (17.8)
		E2ARSEX	17.11% (3.57%)	3770.2 (2.1)	202 (24)

TABLE 5.11 – Résultats en terme de pourcentage de violations, de la consommation énergétique et de la dépense par expérience, pour chaque choix de politique
Moyenne (écart-type)

Nous voyons sur le tableau 5.10, que la politique *E2ARSEC* a tendance à créer 4.4% moins de répliques que *E2ARSEX*. La politique intermédiaire a un nombre de répliques étant entre ces deux politiques extrêmes.

Le tableau 5.11 résume les résultats en termes de violations, de consommation énergétique et de dépense, dans chaque expérience, pour chacun des choix de politique. Ces différences de réplication se remarquent aussi en termes de violations de niveau de service. C'est ainsi, que dans les expériences 1 à faible charge et original, *E2ARSEC* augmente le nombre de violations de 22% et 0.3% respectivement par rapport *E2ARSEX*. Dans l'expérience 2, elle augmente ce nombre de 4.6% et 8.6% respectivement par rapport à *E2ARSEX*. La politique intermédiaire n'étant pas significativement différente en termes de réplications, il en est de même pour le nombre de violations.

Nous comparons à présent ces politiques du point de vue des dépenses et de la consommation énergétique. Dans la moitié supérieure du tableau, nous voyons que l'impact du choix de la politique sur la consommation énergétique est encore plus faible que l'impact du choix du paramètre (partie 5.4.1). L'expérience à faible charge, met en avant cette marginalité de l'impact, avec des différences qui sont inférieures à 1% dans la consommation énergétique. Cependant, dans le cadre de l'expérience originale, la politique *E2ARSEX* consomme et dépense le moins avec une réduction significative de 9.1% et 6.8% respectivement par rapport à *E2ARSEC*.

Un comportement similaire est observé dans la seconde expérience, avec des différences marginales sur la consommation énergétique et plus importante en terme de dépense. Sur la seconde moitié du tableau, nous voyons que *E2ARSEC* a une consommation énergétique quasiment similaire aux autres politiques, avec une augmentation relativement importante de la dépense. À l'inverse, la politique *E2ARSEX* coûte aussi cher que *E2ARS* dans le contexte d'origine, mais réduit le coût de 7.5% dans le contexte à faible charge, avec un impact presque inexistant sur la consommation énergétique.

Nous voyons que le choix de la politique peut avoir une influence sur les performances et la dépense. Elle est cependant presque inexistante en termes de consommation énergétique. En effet, une grande partie de la réduction de consommation énergétique provient de la possibilité d'éteindre des nœuds. Les différences proviennent donc principalement de différences de caractéristiques des centres de données. Il est donc possible que des centres de données ayant plus de disparités en termes de capacité, de coûts, et de puissance amplifient ces impacts. La partie suivante s'intéresse au choix du nombre de nœuds sur lesquels seront stockées les répliques.

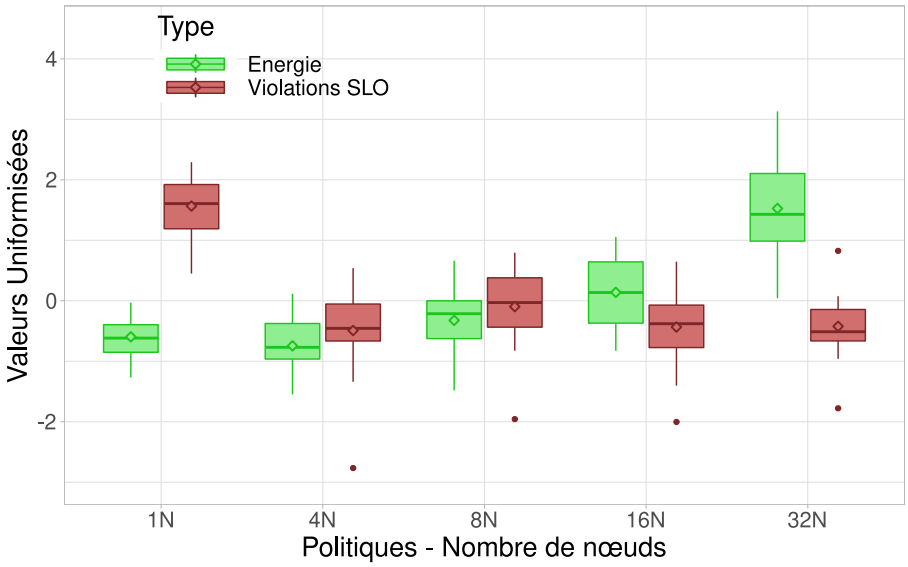


FIGURE 5.2 – Consommation énergétique et nombre de violations par nombre de nœuds allumés

5.4.3 Choix de la proportion de nœuds de stockage

Dans la partie 5.3.3, nous avons émis l'hypothèse que le choix de la proportion de nœuds a un impact à la fois sur la consommation énergétique (en réduisant le nombre de nœuds allumés) et sur les performances (en évitant les goulots d'étranglements). Sachant qu'il y a 64 nœuds par centre de données, nous considérons les cas suivants : un nœud allumé (1N), 4 nœuds (4N), 8 nœuds (8N), 16 nœuds (16N) et 32 nœuds (32N).

Nous comparons ces différentes possibilités du point de vue de la consommation énergétique et du nombre de violations pour voir l'impact sur les performances.

Sur la figure 5.2, nous voyons que la consommation énergétique est la plus faible avec 4 nœuds (4N), même si cette consommation n'est significativement pas différente de la consommation énergétique avec un seul nœud allumé (1N). Cependant, même si leurs consommations énergétiques sont similaires, nous voyons que le passage de 1 à 4 nœuds allumés diminue le nombre moyen de violations qui passe de 102k violations à 88k violations.

Enfin, nous voyons qu'augmenter le nombre de nœuds augmente la consommation énergétique. Alors que le nombre de violations dans les cas 4N, 8N, 16N, 32N, restent similaires. Dans notre contexte, nous gardons l'hypothèse de 4 nœuds allumés, car elle permet de réduire le nombre de violations tout en ayant la consommation énergétique minimum. Il est important de noter, que ce nombre de nœuds dépend de beaucoup de facteurs contextuels tels que les caractéristiques des nœuds, les données à stocker, la durée de location et la fréquence de lecture. Cela permet d'avoir une utilisation des nœuds, représentée sur la figure 5.3.

5.4.4 Évaluation du placement initial proposé

Dans cette partie, nous comparons le placement proposé avec plusieurs stratégies proposées dans la littérature [1].

Pour cela, nous utilisons les expériences présentées dans la partie 4.4 du chapitre 4, avec l'expérience 1 correspondant à une expérience à court terme, avec 150k tâches et l'expérience 2 qui correspond à une expérience plus long terme, avec 2 millions de tâches.

Nous avons comparé notre stratégie à **PEPR** [Tos et al., 2017], **MORM** [Long et al., 2014] et **Boru et al.** [Boru et al., 2015] provenant de la littérature. Nous avons aussi voulu mettre en place

1. Ces stratégies sont aussi utilisées dans la comparaison de la stratégie dynamique (partie 6.5.4 du chapitre 6)

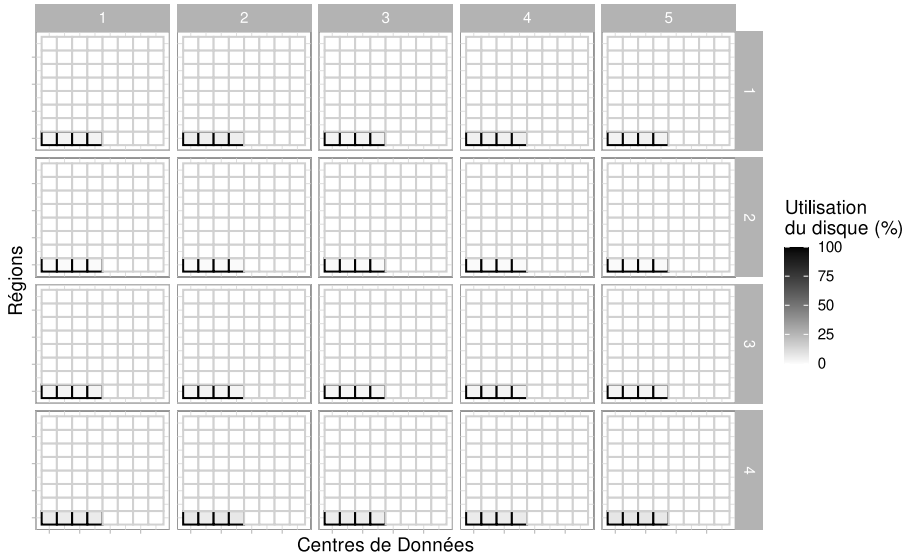


FIGURE 5.3 – Utilisation du disque dur sur tous les nœuds

une stratégie de base qui correspondrait à une stratégie de réplication de données qui crée et place 2 répliques de manière aléatoire. Nous avons nommé cette stratégie de base **3Rand**. Nous faisons un léger rappel du fonctionnement des stratégies qui serviront de comparaison au placement initial.

PEPR : C'est une stratégie de réplication dynamique de données qui se déclenche à chaque violation de SLO, une réplique est créée uniquement si le fournisseur génère du profit. Cette violation de SLO correspond à un temps de réponse supérieur à un seuil correspondant à un objectif de niveau de service. La stratégie réplique au plus proche du nœud sur lequel il y a eu la violation. Un mécanisme de suppression de réplique est mis en place sur la base d'un seuil de nombre de tâches consécutives effectuées sans violation.

Boru : Cette stratégie de réplication dynamique s'appuie sur une architecture hiérarchique pour la mise en place de sa réplication. L'architecture considère 3 niveaux de bases de données : une base de données centrale, une base de donnée par centre de données, et par rack. Ainsi, si le nombre de requêtes dépasse un certain seuil (défini à 30 dans les expériences suivantes) la stratégie réplique la donnée au niveau inférieur si elle consomme moins d'énergie et de bande-passante par rapport au niveau actuel. Afin de pouvoir exécuter les expériences dans un temps raisonnable, nous limitons les temps de transfert de données de cette stratégie. En effet, au vu de son fonctionnement, les temps de transfert ont tendance à atteindre plusieurs heures. Nous avons donc décidé de limiter ce temps à 5 heures, c'est-à-dire que si le temps pour un nœud de récupérer une réplique est supérieur à cette limite, alors on considère que ce temps de transfert est égal à 5h et qu'il n'y aura pas plus d'impact sur le réseau. Cela implique une sous-estimation de nombre de violations et de consommation énergétique car la durée globale de l'expérience sera réduite par cette limite.

MORM : C'est une stratégie statique de réplication de données qui s'appuie sur un algorithme d'optimisation évolutionnaire. La stratégie cherche à minimiser 5 objectifs : la disponibilité, la latence, la consommation énergétique, la charge de travail et le temps de réponse. Elle s'applique avant l'arrivée des premières tâches.

	Paramètres	Répliques	Violations	Énergie (MJ)	Dépense (k$)
XP1	3Rand	512 (0)	63.23% (1.03%)	593.8 (22.3)	19 (0.6)
	MORM	38336 (37702)	0% (0.01%)	501.3 (18.2)	10.3 (9.7)
	Boru	2293 (25)	90.53% (0.42%)	347.1 (0.3)	4.5 (0.02)
	PEPR	78 (8)	58.92% (3.7%)	551.8 (59.1)	22.9 (1.1)
	E2ARS	891 (112)	58.44% (2.49%)	232.5 (38)	13.8 (1.8)
XP2	3Rand	512 (0)	22.97% (1.91%)	7006.3 (63.8)	259.5 (6.5)
	MORM	29886 (10187)	0% (0%)	11570 (5)	14.8 (3.1)
	Boru	4690 (69)	36.25% (0.6%)	3877 (1.2)	53.2 (0.1)
	PEPR	596 (25)	4.35% (0.57%)	4767.9 (0.3)	155.6 (5.4)
	E2ARS	915 (141)	16.69% (3.54%)	3769.9 (2.2)	200.5 (22.4)

TABLE 5.12 – Résultats en terme de nombre de répliques, de pourcentage de violations, de la consommation énergétique et de la dépense par expérience, pour chaque stratégie de réplication
Moyenne (écart-type)

Résultats : Avant de comparer les performances des stratégies, un premier résultat est le temps d'exécution des algorithmes de placement statique. En effet, le temps d'exécution de l'algorithme de MORM dure en moyenne 15 162 secondes contre 41 secondes pour E2ARS.

Les résultats en termes de nombre de répliques à la fin des expériences, nombre de violations, consommation énergétique et dépense sont présentés dans le tableau 5.12.

MORM est une stratégie qui réduit légèrement la consommation énergétique dans la première expérience, avec réduction de 16% par rapport à 3Rand. Cependant, dans la seconde expérience, cette consommation augmente de 65% par rapport à 3Rand. Cela s'explique par le fait que cette stratégie crée beaucoup de répliques, impliquant que beaucoup de nœuds resteront allumés car des données seront stockées dessus malgré qu'aucune tâche ne soit exécutée sur ces nœuds. Cette création de répliques implique un coût de stockage important sur le long terme, avec un coût de transfert de données fixe à l'origine relativement élevé, compensé par un faible coût durant l'exécution. Enfin, cette stratégie réplique en grande quantité sur beaucoup de nœuds, le nombre de violations est particulièrement faible, avec une moyenne de 3.2 violations, impliquant un faible coût de violations.

La seconde stratégie qui consomme le plus dans la première expérience est PEPR. Dans cette expérience, elle réduit la consommation énergétique d'uniquement 7.1% et la réduit de 31.9% dans la seconde expérience par rapport 3Rand. De même, cette stratégie est la seule qui augmente la dépense dans la première expérience avec une augmentation de 20.8%. Comme nous l'avons décrit dans sa présentation, PEPR s'appuie sur les violations pour activer la réplication, et réplique s'il y a du profit. Or, nous voyons que le nombre de répliques reste faible avec un nombre élevé de violations. Cela s'explique par le fait que si les dépenses dépassent les revenus, alors la stratégie arrête de répliquer. Dans les expériences proposées, les revenus sont de 2113$ et 16104$ respectivement pour les expériences 1 et 2, soit un revenu inférieur aux dépenses. Il reste quand même intéressant de constater que PEPR génère moins de violations que 3Rand dans la première expérience, et reste la deuxième stratégie ayant le moins de violations sur la seconde expérience.

Une autre stratégie intéressante est Boru et al., elle est la 2ème stratégie qui dépense le moins dans les 2 expériences, avec une diminution de l'ordre de 70% comparé à 3Rand sur la première expérience. Sur le tableau 5.12, cette stratégie est celle qui a le plus de violations parmi l'ensemble des stratégies. Cela s'explique dans un premier temps par l'architecture de Boru qui est hiérarchique, et génère donc des goulots d'étranglements au niveau des bases de données, ce qui augmente le nombre de violations. Cette stratégie va créer beaucoup de répliques, et va favoriser la création de répliques dans chaque région, ce qui permet de réduire les coûts de transfert. Comme nous l'avons explicité lors de la présentation des stratégies, nous avons limité son temps de transfert, cela implique une sous-estimation de la consommation énergétique.

Enfin, nous voyons qu'avec une augmentation de 70% du nombre de répliques par rapport à 3Rand, E2ARS arrive à la fois à réduire la consommation énergétique de 61% et la dépense de 27.5% pour l'expérience 1. Ces réductions s'amoindrissent avec l'expérience 2, avec une réduction de 47.2% de la consommation énergétique et de 22.7% de la dépense. Ainsi, comparé à MORM, nous réussissons, à réduire la consommation énergétique et la dépense par rapport à 3Rand. Cependant, nous réduisons que faiblement le nombre de violations par rapport à 3Rand, ce qui est expliqué par le faible nombre de répliques créées. Ce problème sera ainsi étudié dans le chapitre suivant qui introduira la partie dynamique de la stratégie permettant d'adapter le nombre de répliques pour améliorer les performances.

5.5 Conclusion

Dans ce chapitre, nous avons présenté le placement initial (E2ARS) de la stratégie de réplication de données. Ce placement se fait en 2 étapes afin de réduire l'espace de recherche. Dans la première étape, le choix concerne la réplication inter-centres de données. L'algorithme d'optimisation présenté dans cette étape, s'appuie avant tout sur l'hétérogénéité entre les centres pour créer et placer les répliques dans les centres de données. La seconde étape place les répliques, issues de l'étape précédente, sur les nœuds dans chaque centre de données, en s'appuyant sur la consolidation et les états de veilles. Cela permet de concentrer les données sur un petit nombre de nœuds, afin d'éteindre les autres nœuds qui ne possèdent pas de données.

Nous avons vu dans l'évaluation que E2ARS permettait de réduire à la fois la consommation énergétique et la dépense par rapport à la stratégie de base. Cela n'est pas le cas d'autres stratégies de réplication de données qui sont moins efficaces pour réduire la consommation énergétique, et plus souvent efficaces pour réduire la dépense. Enfin, nous avons vu que du point de vue des performances, le placement proposé est légèrement plus efficace que le placement aléatoire de trois répliques . Durant l'évaluation, nous avons pu estimer l'impact que pouvait avoir les administrateurs et administratrices sur la réplication.

En effet, nous avons vu que ces derniers et dernières peuvent être impliqués de différentes manières dans la réplication. Pouvant faire des hypothèses de charge de travail et de temps de location, permettant de créer plus où moins de répliques selon les besoins de l'architecture. Choisissant aussi, de mettre en place une politique plus écoresponsable en réduisant la consommation énergétique, ou appliquer une politique qui s'oriente plus vers une réduction de la dépense. Enfin, ces administratrices et administrateurs peuvent aussi définir le pourcentage de nœuds à allumer, répondant ainsi aux compromis entre réduire la consommation énergétique et éviter les goulots d'étranglement.

Le chapitre suivant introduit la partie dynamique de la stratégie proposée de réplication de données. Cette réplication dynamique s'applique à la suite de ce placement initial, qui sera considéré comme la réplication de base. L'idée de cette partie dynamique sera d'augmenter le nombre de répliques en cas de violations, et de supprimer des répliques s'il n'y en a plus. Cependant, elle ne supprimera pas les répliques créées durant le placement statique, mais uniquement les répliques créées durant la partie dynamique.

Chapitre 6

Stratégie dynamique de réplication de données

Sommaire

6.1 Déroulé de la stratégie

Dans ce chapitre, nous allons décrire la partie dynamique de la stratégie de réplication de données. L'ensemble de la stratégie est nommée *Dynamic Energy and Expenditure Aware Replication Strategy* ou *DE2ARS*. La figure 6.1 décrit l'ensemble des étapes de la stratégie DE2ARS proposée dans ce manuscrit.

Le placement initial (à gauche) des répliques a été introduit dans le chapitre précédent. Puis, les espaces de stockage pour la carte de contrôle, la récupération d'informations et la planification de la réplication sont alloués.

Suite à cette mise en place, à gauche sur la figure, les requêtes utilisateurs arrivent dans les centres de données. Elles sont placées sur des nœuds pour être exécutées. Comme ce sont les premières requêtes, aucune réplication n'est prévue à cette étape.

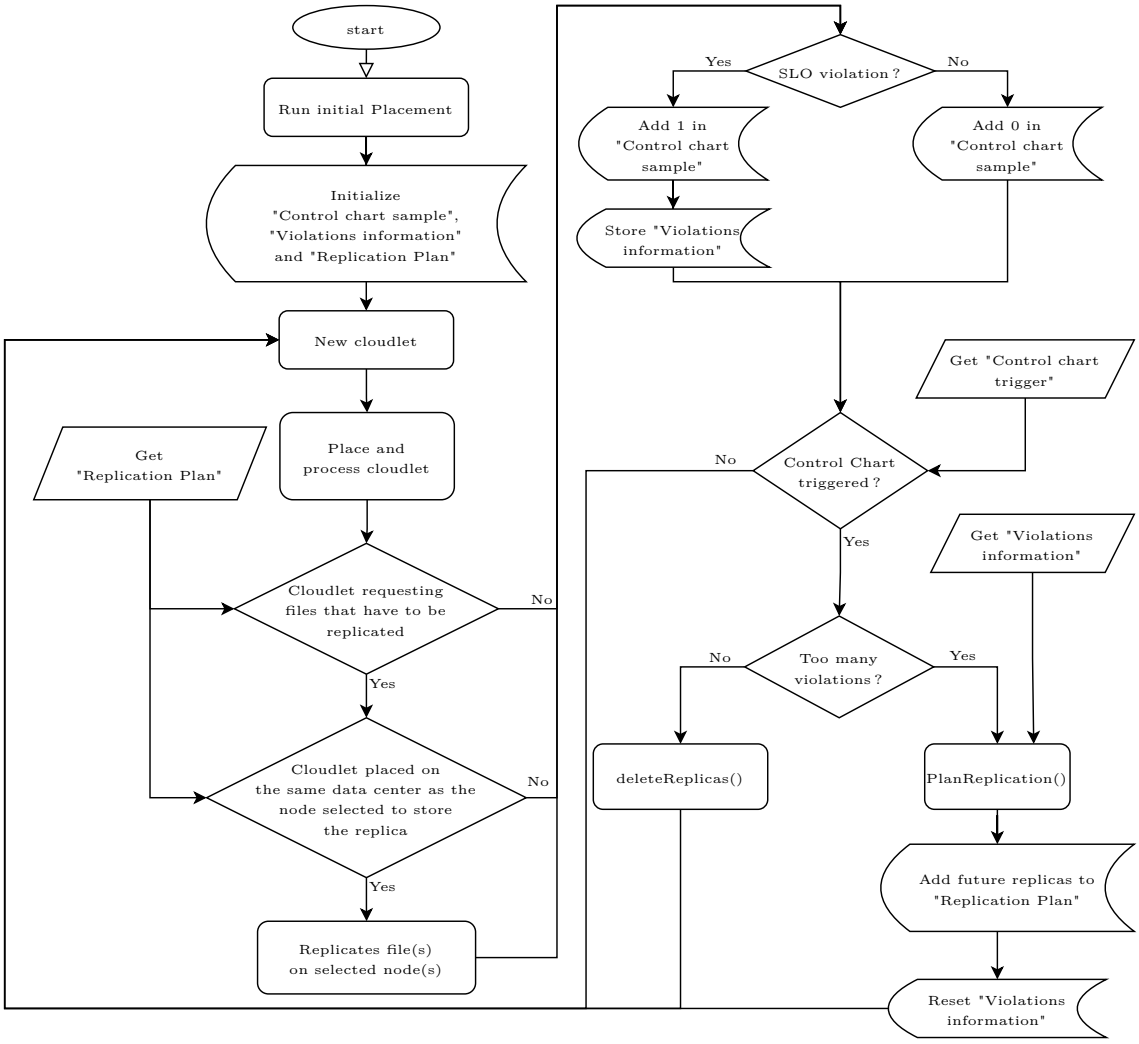


FIGURE 6.1 – Fonctionnement global de la stratégie de réplication

Sur la partie droite de la figure sont présentées les étapes principales de la stratégie de réplication. Dans un premier temps, nous abordons la méthode de détection du moment de réplication. Elle est mise en place au travers d'une carte de contrôle décrite dans la partie 6.2. Comme cela sera détaillé dans la partie 6.3.1, lorsque la requête génère une violation, plusieurs informations liées à cette violation sont stockées. Lorsque la carte lève des alertes, (*Control Chart Triggered* dans la figure) il existe deux possibilités.

Soit, il y a trop de violations, nécessitant une augmentation du nombre de répliques. Cette augmentation, présentée dans la partie 6.3, va planifier les répliques à créer et les nœuds sur lesquels elles seront stockées, avec la fonction *PlanReplication()*. Cette planification sera stockée dans "Replication Plan" envoyé aux centres de données.

Soit, il y a trop peu de violations, nécessitant donc une baisse du nombre de répliques présentée dans la partie 6.4.

Lorsque cette partie de la stratégie est terminée, de nouvelles tâches sont arrivées. Si l'une de ces tâches nécessite une donnée à répliquer, et qu'elle est exécutée dans le centre de données sur lequel cette donnée doit être répliquée, alors la stratégie profite du transfert pour exécuter la tâche, pour répliquer après cette dernière.

Après avoir décrit le fonctionnement de cette stratégie, nous évaluerons l'impact de différents choix proposés durant ce chapitre et l'ensemble de la stratégie par rapport à d'autres stratégies de la littérature. Cette évaluation sera présentée dans la partie 6.5. S'ensuivra dans la partie 6.6, une discussion autour des résultats obtenus.

6.2 Détection du moment de la réplication

6.2.1 Déclenchements

Dans la stratégie de réplication que nous proposons, on doit tenir compte de deux évènements : une montée de charge de travail et une baisse de charge de travail. Ces évènements peuvent se traduire de différentes manières avec, par exemple, une augmentation du temps de réponse ou encore augmentation du taux de requêtes.

Dans notre contexte, le temps de réponse d'une requête doit satisfaire les objectifs de niveau de service du fournisseur. Ainsi, s'il y a une violation de cet objectif de niveau de service, cela signifie que le temps de réponse est trop élevé. Ce temps de réponse dépend à la fois du temps de transfert de données, mais aussi du temps d'exécution de la tâche.

Ces violations ont un impact économique sur le fournisseur de Cloud, qui correspond à un pourcentage du prix de location. Lorsque la proportion de violations dépasse certains seuils, le pourcentage de remboursement augmente.

Nous nous appuyons donc sur le nombre de violations pour détecter les changements de charges de travail. Ce nombre sera donc en entrée de la carte de contrôle sur laquelle on s'appuie pour la réplication et la suppression de répliques.

6.2.2 Utilisation des cartes de contrôle

Comme nous l'avons décrit dans la partie 3.3.2 du chapitre 3, les cartes de contrôle se construisent en se basant sur une ligne centrale qui sert de cible, des limites de chaque côté de la cible et des zones entre la cible et les limites.

Nous nous appuyons ici sur une carte p, c'est-à-dire une carte qui se base sur la proportion de "produits défectueux", ici la proportion de violations sur le nombre de tâches exécutées. Ces cartes correspondent à une loi Binomiale $\mathcal{B}(n, p)$, avec n la taille de l'échantillon et p la probabilité de succès/échec, ici la probabilité de violations.

Nous supposons dans la suite de ce chapitre, qu'au-delà de 5% de violations, les coûts deviennent trop important pour le fournisseur. Cela correspond à une limite que l'on ne doit pas dépasser. La loi Binomiale étant proche de la symétrie selon la taille de l'échantillon, nous pouvons considérer une cible à une valeur de 2.5%.

Avoir un paramètre p avec des valeurs trop faibles ou trop élevés entraîne un problème de taille d'échantillon. Si la taille de l'échantillon est trop faible, il sera alors difficile de différencier deux proportions proches. Cette faible différenciation augmenterait le nombre de fausses alertes, et ne rendrait pas compte de cette évolution de la charge de travail. Pour répondre à cette situation, quelques travaux dans la littérature proposent différentes règles pour estimer la taille minimale d'un échantillon. Dans ce contexte, [Schader and Schmid, 1989] compare deux de ces règles :

$$n * p * (1 - p) > 9 \Leftrightarrow n > \frac{9}{0.025 * 0.975} \Leftrightarrow n > 369.2 \tag{6.1}$$

et

$$\left.\begin{array}{l} n * p > 5, \quad 0 < p \leq 0.5 \\ n * (1 - p) > 5, \quad 0.5 \leq p < 1 \end{array}\right\} \Leftrightarrow n > \frac{5}{0.025} \Leftrightarrow n > 200 \tag{6.2}$$

Comme nous le voyons ci-dessus, la règle 6.1 est la plus contraignante et sera donc appliquée dans la suite de ce chapitre. Ainsi, pour une valeur $p = 0.025$, l'échantillon doit avoir une taille strictement supérieure à 369.2, et sera de ce fait de taille $n = 370$. Les limites supérieure (UCL) et inférieure (LCL) de contrôle utilisées sont donc les suivantes :

$$UCL = p + 3 * \sqrt{\frac{p(1-p)}{n}} \Leftrightarrow UCL = 4.93\% \tag{6.3}$$

$$LCL = p - 3 * \sqrt{\frac{p(1-p)}{n}} \Leftrightarrow LCL = 0.07\% \tag{6.4}$$

Comme décrit dans la présentation des cartes de contrôle la mise en place des règles nécessitent des zones autour de la cible qui s'appuient sur la variance de la loi utilisée. Dans notre cas, les zones auxquelles appartiennent les échantillons sont définies de la manière suivante :

$$zone\, 3\sigma \in [0.07\%, 4.93\%] \tag{6.5}$$

$$zone\, 2\sigma \in [0.88\%, 4.12\%] \tag{6.6}$$

$$zone\, 1\sigma \in [1.68\%, 3.31\%] \tag{6.7}$$

De plus, il est nécessaire de mettre en place des règles qui déclenchent des alertes. Nous avons adapté les règles présentées dans la partie 3.3.2 à notre contexte. Ici, nous nous concentrons sur les règles qui permettent d'éviter le dépassement de 5% de violations. Afin de simplifier la compréhension des règles mises en place, nous les avons définies puis remises dans notre contexte, et enfin nous les avons appliquées sur les exemples présentés dans les figures 6.2. Nous appliquons donc les règles suivantes :

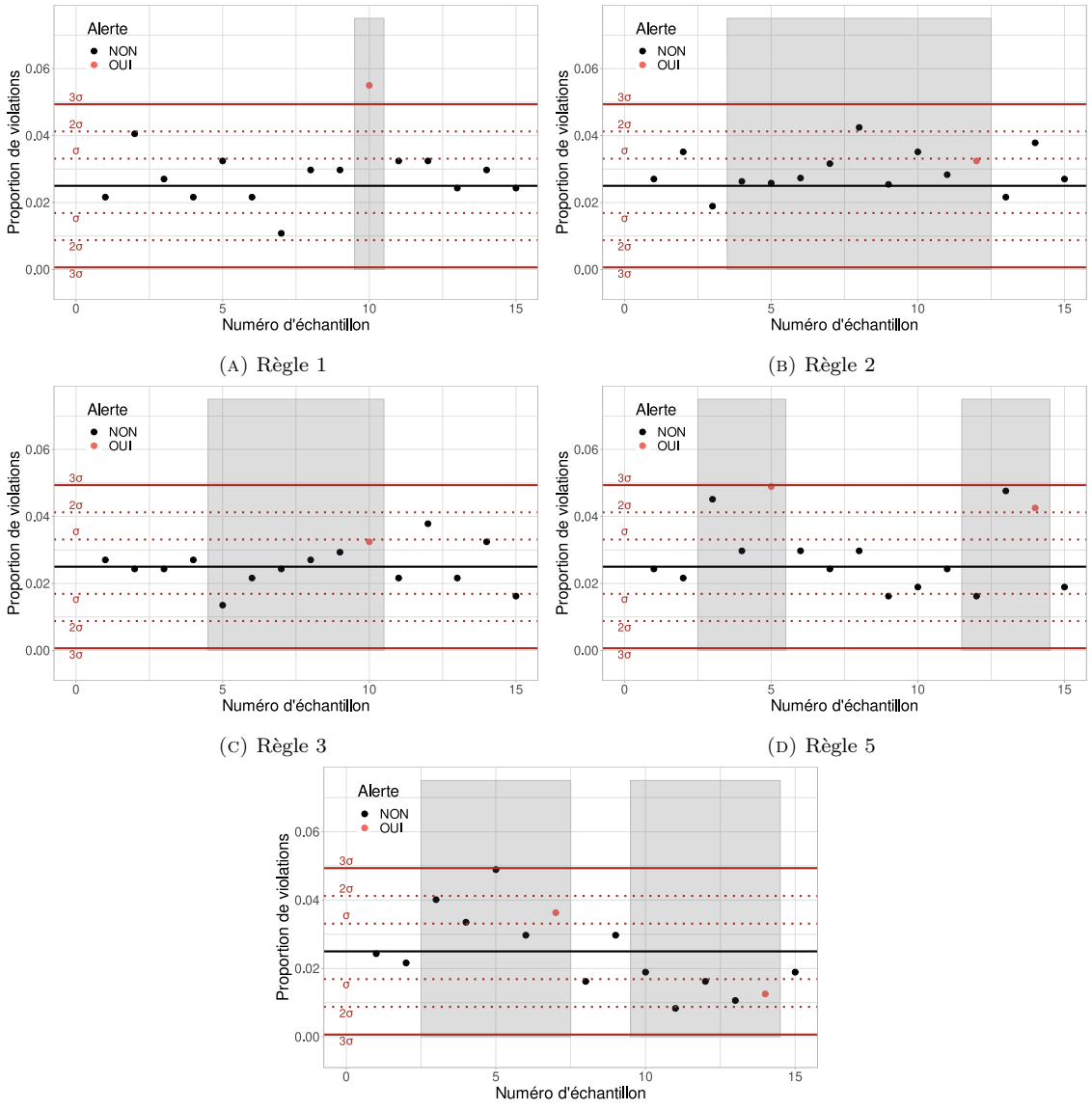
Règle 1. **1 point au-delà de la zone 3σ**
Cette règle sera activée lorsque la proportion de violations dépasse 4.93% ou est inférieure à 0.07% (figure 6.2a)

Règle 2. **9 points ou plus qui sont du même coté de la moyenne**
Elle sera activée, si au moins 9 échantillons à la suite ont une proportion de violations supérieure ou inférieure à 2.5% (figure 6.2b)

Règle 3. **6 points ou plus qui augmentent (/diminuent) continuellement**
Comme sur la figure 6.2c, 5 échantillons en continu ont une proportion supérieure (/inférieure) à la précédente

Règle 5. **Au moins 2 points sur 3 qui sont au-delà de 2σ dans la même direction**
Cette règle s'active lorsqu'au moins 2 échantillons parmi 3 à la suite ont une proportion de violations supérieure à 4.1% ou, inférieure à 0.88% (figure 6.2d)

Règle 6. **Au moins 4 points sur 5 qui sont au-delà de 1σ dans la même direction**
Cette dernière s'active, lorsque 4 échantillons parmi 5 à la suite ont une proportion de violations supérieure à 3.3% ou inférieure 1.69% (figure 6.2e)



(A) Règle 1

(B) Règle 2

(C) Règle 3

(D) Règle 5

(E) Règle 6

FIGURE 6.2 – Exemples d'échantillons qui lèvent des alertes selon différentes règles

Les autres règles correspondent au fait qu'il y a trop de points dans la zone 1σ (règle 7) ou au-delà de 1σ (règle 8), ce qui correspond plus à des déviations de la dispersion, et non d'une déviation de la moyenne. Et la dernière règle, représente une alternance des points (règle 4), qui ne permet pas de répondre à notre problème.

Un exemple d'application de chaque règle est présenté sur la figure 6.2. Sur ces figures, les zones grises correspondent aux points d'intérêt et les points rouges, sont les échantillons qui lèvent des alertes.

6.2.3 Mise en place de la carte de contrôle

Dans la section précédente, les cartes de contrôle génèrent des alertes qui s'appuient sur des règles. Ces alertes permettent soit d'activer la réplication, soit supprimer des répliques. Pour faire cette différenciation, nous avons mis en place des scores pour chaque alerte.

Avant l'évaluation de l'échantillon au travers des règles, on donne à cet échantillon un score nul. Lorsque l'alerte montre une augmentation de la charge de travail, on ajoute $+1$ à ce score. A l'inverse, lorsque l'alerte montre une diminution de la charge de travail, on ajoute -1 à ce score.

C'est-à-dire que pour les règles 1,2,5 et 6, on regarde sur quel coté de la moyenne la règle s'active. Ainsi, si pour ces règles, l'ensemble des échantillons associés ont une proportion qui est au-dessus de la moyenne cible, alors l'alerte correspond à une augmentation de la charge de travail, et vice versa.

Pour la règle 3, on regarde l'évolution de la proportion de violations pour l'ensemble des échantillons associés. Si cette proportion augmente, alors il y a une augmentation de la charge, sinon la charge baisse.

A la fin de cette évaluation, si le score est positif, alors la stratégie va créer des répliques. Si le score est nul alors la stratégie ne s'active pas. Enfin, si le score est négatif alors elle applique la suppression des répliques.

Variable	Type	Description
CC	**Carte de contrôle**	Carte de contrôle
ech	List d'entier	Échantillon en cours de la carte de contrôle, chaque élément correspond à une tâche, et prend la valeur 1 ou 0 selon qu'il génère une violation ou non
file	**Donnée**	Donnée nécessaire à l'exécution de la tâche
InfoSlo	Dictionnaire d'entier : Liste de **Nœuds**	Dictionnaire qui prend comme clé le numéro de la donnée et comme valeur les nœuds sur lesquels il y a eu une violation
propBreach	Flottant	Proportion de violations de SLO sur l'échantillon
score	Entier	Score à l'issue de l'activation de la carte
t_SLO	Flottant	Temps d'exécution limite prévu par le SLO
task	**Tâche**	Tâche terminée

TABLE 6.1 – Description des variables utilisées dans l'algorithme 4

Pour faciliter la compréhension de l'algorithme 4, le tableau 6.1 présente les variables utilisées dans cet algorithme. Certaines variables correspondent à des objets présentés dans la partie 5.3. C'est le cas des **nœuds**, qui possèdent un numéro d'identification (idN), une capacité de stockage (cp), l'identifiant du centre de données dans lequel il est (DCid) et enfin différentes informations associées au nœud (infos). De même pour les **données** identifiées pas un numéro (idF), avec une taille en Mo (size) et qui est stockée avant la réplication sur un nœud d'origine, dont on connaît l'identifiant (origin).

De nouveaux objets sont introduits dans cette partie comme la **tâche**, identifiée par un numéro de tâche (idT). Une tâche a besoin de fichiers pour être exécutée (files), et est exécutée sur un nœud

Objet	Variable : Type	Description
Carte de contrôle	target : Flottant	Cible de la carte de contrôle
	limits : Liste de Flottant	Liste des limites de chaque zone
	rules : Liste d'Entier	Liste d'entier des règles actives
	echSize : Entier	Taille d'un échantillon
	hist : Liste de Flottant	Historique des proportions de violations
	isTriggered : Booléen	La carte de contrôle lève des alertes
	score : entier	Score de la carte de contrôle, lié aux alertes
Donnée	idF : Entier	Identifiant de la donnée
	size : Entier	Taille de la donnée
	origin : Entier	Identifiant du nœud qui stocke la donnée avant le placement inital
Nœud	idN : Entier	Identifiant du nœud
	cp : Entier	Capacité de stockage
	DCid : Entier	Identifiant du centre dans lequel il est
	infos : Texte	Différentes informations associées au nœud
Tâche	idT : Entier	Identifiant de la tâche
	files : Liste de **Données**	Liste des fichiers nécessaires à l'exécution de la tâche
	tps : Flottant	Temps d'exécution de la tâche
	node : **Nœud**	Nœud qui exécute la tâche

Table 6.2 – Description des objets de l'algorithme 4

(node). Le temps de réponse de la tâche, comprenant le temps de transfert des fichiers et le temps d'exécution est noté tps.

Nous introduisons enfin la méthode de déclenchement de la stratégie avec la carte de contrôle. Comme nous l'avons présenté précédemment, une **carte** correspond à une cible (target), avec des zones autour (limits) ainsi qu'un ensemble de règles à appliquer (rules). Pour activer les règles, les cartes ont un historique des moyennes des échantillons, correspondant ici à la proportion de violations (hist). Dans notre cas, la taille de l'échantillon de la carte de contrôle est fixe (echSize). Enfin, lorsque des règles sont activées, la carte lève des alertes (isTriggered), et ces alertes sont associées à un score (score) pour savoir s'il y a trop, ou pas assez de violations. Un résumé des objets utilisés est présenté dans le tableau 6.2.

L'algorithme 4 résume le fonctionnement de la détection du moment de réplication. Dans un premier temps, nous vérifions à chaque tâche si elle génère une violation de SLO, et stockons l'information dans l'échantillon (lignes 1 - 8). De plus, lorsqu'il y a une violation, nous stockons les informations de la tâche (lignes 3 - 5).

Lorsque la taille de l'échantillon atteint la taille prévue par la carte, nous calculons la proportion de violations de cet échantillon et vérifions si la carte lève une ou plusieurs alertes. En se basant sur le score des alertes de la carte, (i) nous créons des répliques et vidons les informations liées aux violations (ligne 14), ou (ii) supprimons des répliques (ligne 18).

À la fin de cette étape, nous vidons l'échantillon pour commencer un nouveaux cycle de récupération d'informations (ligne 21).

6.3 Réplication de données

Dans cette partie, nous allons présenter les étapes de la création de répliques lorsque la carte lève une alerte associée à une augmentation de charge. Cette étape implique une récupération d'informations tout au long de l'exécution, permettant de répondre aux questions de quelles données répliquées et où placer les répliques.

Procedure 4 Déclenchement de la création ou suppression de répliques

Input: task : tâche terminée
Input: CC : Carte de contrôle (construite)
Input: ech : échantillon de la carte
Input: InfoSlo : Dictionnaire dont chaque clé correspond à une donnée et la valeur est une liste de nœuds qui correspondent à des violations de SLO
Output: ech
Output: InfoSlo
Output: CC

```
1: if task.tps > t_SLO then
2:    ech.add(1)
3:    for file in t.files do
4:       InfoSlo.get(file.idF).add(task.node)
5:    end for {Ajoute 1 à l'échantillon de la carte, et les informations de la violation de SLO}
6: else
7:    ech.add(0) {Ajoute 0 à l'échantillon de la carte}
8: end if
9: if CC.echSize == size(ech) then
10:   propBreach = mean(ech)
11:   CC.hist.add(propBreach)
12:   if CC.isTriggered then
13:      if CC.score > 0 then
14:         planReplication(InfoSlo)
15:         InfoSlo = empty() {Vider les informations des violations}
16:      end if
17:      if CC.score < 0 then
18:         deleteReplicas()
19:      end if
20:   end if
21:   ech = empty() {Vide l'échantillon}
22: end if
```

6.3.1 Récupération des informations

Comme pour les cartes de contrôle, nous créons un espace pour stocker les informations des violations à la suite du placement statique. Ainsi, durant l'exécution, lorsqu'une tâche génère une violation, en plus d'ajouter une valeur à l'échantillon de la carte de contrôle, elle envoie aussi les informations associées à la violation. Ces informations contiennent principalement la donnée requêtée et le nœud sur lequel la tâche est exécutée.

Cela permet d'obtenir des informations sur les données qui génèrent le plus de violations et des nœuds qui sont les plus problématiques pour l'exécution de requêtes sur cette donnée. Nous allons ensuite présenter comment sont utilisées les informations récupérées durant l'exécution.

6.3.2 Choix des données à répliquer

Lorsque la carte de contrôle génère une alerte, la partie dynamique de la réplication de données se met en place. Dans cette partie, nous étudions le cas où l'alerte montre qu'il y a trop de violations à partir des cartes de contrôle, et qu'il est nécessaire de créer de nouvelles répliques. Dans ce contexte, nous nous basons sur les informations de violations récupérées pour choisir la donnée à répliquer.

Pour cela, les données sont triées par nombre de violations et nous sélectionnons les données correspondant à une grande partie des violations. Cette sélection s'appuie sur le principe de Pareto qui nous permet de supposer que 80% des violations proviennent de 20% des données. Nous sélectionnons les données qui correspondent à 80% des violations, ce seront les données à répliquer.

6.3.3 Placement des répliques

De même, que pour le choix de la donnée à répliquer, nous appliquons le même raisonnement sur les régions et sur les centres de données. Sur la base du nombre de violations de chaque donnée, on récupère les régions qui correspondent à 80% des violations de la donnée. Pour chaque région, on récupère les centres de données qui correspondent à 80% des violations de la région pour chaque donnée.

À la fin de cette étape, nous savons quelles données sont à répliquer et dans quels centres de données. Le choix du nœud sur lequel la réplication se fera correspond à un nouveau compromis entre les problématiques énergétiques et les performances. En effet, soit la réplication se fait sur un nœud qui stocke déjà des données afin de garder les autres nœuds éteints. Cela aura un impact sur les performances, car cela peut générer un goulot d'étranglement. À l'inverse, il est possible de favoriser la réplication sur un nœud qui ne stocke pas de données, pour améliorer l'équilibrage de charge, augmentant le nombre de nœuds allumés.

Pour répondre à ce compromis, nous proposons de mettre en place un score lié à l'utilisation de la bande-passante. Si le temps de transfert par Méga-octet (Mo) est supérieur à un certain seuil alors on considère le nœud comme un goulot d'étranglement. Ce seuil peut être défini par l'administrateur et l'administratrice pour mettre en place une politique plus ou moins énergivore. Pour la suite, nous considérons un seuil de 0.005 s/Mo pour chaque nœud. Une évaluation des différentes possibilités de seuil est étudiée par la suite dans la partie 6.5.1.

Sur cette base, nous envoyons l'information des futures répliques à créer à chaque centre de données, pour qu'elles s'appuient sur les requêtes en cours et futures pour répliquer.

6.3.4 Création des répliques

L'idée est d'éviter que la réplication ait un coût supplémentaire, et de s'appuyer sur les transferts associés aux tâches pour activer la réplication.

Lorsque les centres de données reçoivent les informations des futures répliques à créer et les nœuds cibles, les centres vérifient si les tâches contiennent les données nécessitant la réplication. Si c'est le cas, lorsque la tâche se termine, la donnée est répliquée sur le nœud cible. Afin d'éviter

d'augmenter le nombre de transferts, le centre de données vérifie à chaque tâche les données auxquelles elle doit avoir accès. Ainsi, si une des données de la tâche est à répliquer, alors on profite du transfert associé à la tâche, pour répliquer sur le nœud cible à la fin de cette dernière.

Nous allons à présent décrire la deuxième partie de la réplication dynamique : la suppression de répliques.

6.4 Suppression de répliques

Une gestion élastique des resources implique la possibilité d'ajuster le nombre de répliques lorsqu'il y a une stabilisation voire une baisse de la charge de travail en s'appuyant sur la carte de contrôle. Lorsque la charge de travail diminue, il y a moins de violations activant la carte de contrôle. Elle s'ajoute à la réplication de la montée en charge qui reste active. Comme pour l'augmentation de la charge, la suppression est déclenchée par la carte de contrôle sur les violations. Ainsi, s'il y a une baisse ou trop peu de violations, alors il est possible de supprimer des répliques.

6.4.1 Choix du nombre de répliques

Nous nous sommes intéressés au nombre de répliques à supprimer. Nous avons testé plusieurs possibilités qui peuvent avoir leurs avantages et inconvénients. Ces essais sont représentés dans la partie 6.5.2.

Une première possibilité étudiée a été de considérer uniquement la suppression réplique par réplique. Cela permet de diminuer lentement le nombre de répliques, évitant le sous-provisionnement si la charge augmente. Cette vitesse de suppression ne permet pas d'adapter efficacement l'extinction des nœuds, car s'il y a beaucoup de répliques, alors le temps pour supprimer assez de répliques afin de pouvoir éteindre un nœud peut être long.

A l'inverse, nous avons testé la possibilité de supprimer toutes les nouvelles répliques créées en même temps. Comme nous le verrons dans la section d'évaluation des performances (partie 6.5.2), cela à l'effet inverse car les besoins en répliques peuvent rester et donc lever de nouvelles alertes. Cela engendre une alternance entre la création de répliques pour répondre aux violations et la suppression de répliques lorsque l'on atteint notre objectif en termes de violations.

Nous avons donc choisi d'utiliser une proportion de répliques à supprimer, afin d'accélérer la suppression lors de la baisse de charge, sans en supprimer trop pour éviter les alternances entre suppression et réplication. Par la suite, nous considérons une proportion de 10% de suppression de nouvelles répliques.

6.4.2 Choix des répliques à supprimer

Enfin, nous nous sommes intéressés à la manière de choisir les répliques à supprimer. Pour cela, nous avons mis en place un score de suppression basé sur la consommation énergétique et la dépense.

Ces scores pourraient correspondre à des fonctions objectifs dans le cadre d'une optimisation multi-objectifs. Il aurait été possible de mettre en place des méta-heuristiques comme *SPEA2* présentée dans le chapitre précédent. Or les algorithmes d'optimisation multi-objectif peuvent avoir un temps d'exécution élevé et être coûteux en terme de consommation énergétique. De plus, la fréquence d'activation de la carte de contrôle à la suite d'une forte charge peut être élevée et donc la fréquence d'utilisation d'un algorithme d'optimisation pourrait rendre cela contre-productif.

Nous avons ainsi opté pour un calcul de score pour chaque donnée et un tri de ces données selon ce score pour mettre en avant les données à supprimer. Ce score est calculé en s'appuyant sur les modèles 3.7, correspondant à la consommation énergétique statique, et 3.16, correspondant à la dépense du stockage, présentés dans le chapitre 3.

Pour des raisons de facilité et de rapidité, nous nous sommes concentrés sur la consommation énergétique et le coût du stockage uniquement.

Dans le calcul de ce score, nous introduisons la fonction $time_{read}(f_i, n_j)$, qui retourne l'horodatage de la dernière requête pour la réplique de la donnée f_i stockée sur le nœud n_j, et la variable *now* qui correspond à l'horodatage du moment où le score est calculé. Enfin, une dernière notation introduite est *propEC*, le poids associé à la partie consommation énergétique. Les scores de consommation énergétique et de dépense sont les suivants :

$$EC_{score}(f_i, n_j) = \phi(f_i, n_j) * EC_{static}(\emptyset, n_j, diff_{time}(f_i, n_j)) * \frac{s(f_i)}{\sum_{f'_i \in F} \phi(f'_i, n_j) * s(f'_i)} \tag{6.8}$$

$$EX_{score}(f_i, n_j) = \phi(f_i, n_j) * EX_{\text{Storage}}(n_j, f_i, diff_{time}(f_i, n_j)) \tag{6.9}$$

$$avec \quad : diff_{time}(f_i, n_j) = now - time_{read}(f_i, n_j) \tag{6.10}$$

Pour les rendre comparables, ces scores sont centrés et réduits. C'est-à-dire que l'on soustrait à ces scores, la moyenne des scores, et que l'on divise le résultat par l'écart-type des scores. Cela permet de les mettre à la même échelle tout en gardant la distribution des scores et de mettre en avant les scores particulièrement élevés pour la consommation énergétique ou pour la dépense. Ce calcul se fait de la manière suivante avec $AC \in \{EC, EX\}$, $\overline{AC_{score}}$ la moyenne des scores, et $Var(AC_{score})$ la variance des scores :

$$CR_{AC}(f_i, n_j) = \frac{AC_{score}(f_i, n_j) - \overline{AC_{score}}}{\sqrt{Var(AC_{score})}} \tag{6.11}$$

$$avec : \overline{AC_{score}} = \frac{\sum_{i'=1}^{z} \sum_{j'=1}^{m} AC_{store}(f_{i'}, n_{j'})}{\sum_{i'=1}^{z} \sum_{j'=1}^{m} \phi(f_{i'}, n_{j'})} \tag{6.12}$$

$$Var(AC_{score}) = \frac{\sum_{i'=1}^{z} \sum_{j'=1}^{m} [AC_{store}(f_{i'}, n_{j'}) - \overline{AC_{store}}]^2}{\sum_{i'=1}^{z} \sum_{j'=1}^{m} \phi(f_{i'}, n_{j'})} \tag{6.13}$$

Le score de suppression est donc calculé de la manière suivante :

$$DelScore(f_i, n_j) = CR_{EC} * propEC + CR_{EX} * (1 - propEC) \tag{6.14}$$

Puis, on trie les répliques selon ce score et on supprime les répliques avec les scores les plus élevés. Les administrateurs et administratrices auront ainsi la possibilité de favoriser les suppressions qui réduisent la consommation énergétique ou les suppressions qui réduisent la dépense.

Il est à noter cependant, que nous ne considérons ici que les coûts énergétiques et monétaires associés au stockage. Cela peut être approfondi pour prendre en compte les coûts de transfert.

6.5 Évaluation des performances de la stratégie de réplication

Dans cette partie, nous allons évaluer les performances de DE2ARS, la stratégie proposée. D'un coté, nous allons évaluer les performances de la stratégie en prenant en compte différentes configurations. De l'autre, considérant une unique variante, l'évaluation par rapport à d'autres stratégies de la littérature.

Pour évaluer les performances de la stratégie de réplication, nous nous appuyons sur les mêmes expériences du chapitre précédent présentées dans la partie 4.4 du chapitre 4. Nous utilisons donc deux expériences : (i) l'expérience 1 (court terme), qui simule 4h d'exécution et 150k tâches et (ii) l'expérience 2 (long terme), qui simule 4 jours d'exécution pour 2 millions de tâches.

Nous allons dans un premier temps étudier l'impact du seuil de limite de bande-passante présentée dans la partie 6.3.3, lors du choix du nœud dans le placement de la réplique.

Nous étudierons ensuite l'impact des différentes possibilités de nombre de répliques supprimées (partie 6.5.2), l'impact des différents poids dans le score de suppression (partie 6.5.3).

Nous terminerons par la comparaison des performances de la stratégie de réplication de données proposée, aux performances d'autres stratégies issues de la littérature (partie 6.5.4).

6.5.1 Comparaison des seuils d'utilisation de bande-passante

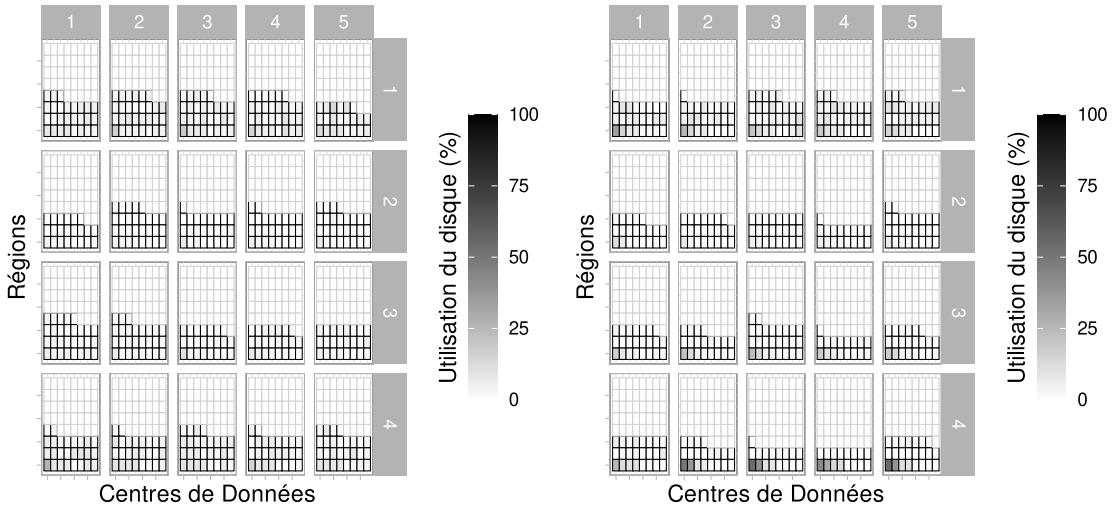
Nous allons comparer les différentes possibilités de seuils d'utilisation de bande-passante présentés dans la partie 6.3.3. Ces seuils correspondent à de la bande-passante disponible en *upload*. Nous comparons ici plusieurs possibilités avec d'un coté des seuils faibles de 0.0001 et 0.001 s/Mo, nommés *Very Low* et *Low* respectivement. A l'inverse, des seuils plus élevés sont aussi comparés avec un seuil *Very High* de 0.1 s/Mo et un seuil *High* de 0.01 s/Mo. Enfin, un seuil intermédiaire noté *Mid* de 0.005 s/Mo.

	Paramètres	Réplications	Violations	Énergie (MJ)	Dépense (k$)
XP1	Very Low	5542 (161.5)	4.12% (0.18%)	218.2 (2.9)	4.23 (0.17)
	Low	5508 (247.3)	4.09% (0.3%)	214.8 (3.6)	4.12 (0.25)
	Mid	5552 (215)	4.14% (0.27%)	209.7 (3.9)	4.23 (0.23)
	High	5465 (233.5)	4.04% (0.27%)	202.9 (4)	4.27 (0.23)
	Very High	5431 (280.5)	4.05% (0.33%)	186.5 (3.6)	4.24 (0.27)
XP2	Very Low	2121 (127.4)	0.166% (0.015%)	4588 (84.2)	56.96 (4.23)
	Low	2101 (114)	0.171% (0.012%)	4255 (65.3)	56.53 (4.03)
	Mid	2067 (121.2)	0.166% (0.019%)	4159 (52.9)	56.59 (4.31)
	High	2048 (127.2)	0.184% (0.022%)	4128 (70.3)	59.76 (5.24)
	Very High	2121 (132.1)	0.17% (0.02%)	3833 (28)	56.49 (4.88)

TABLE 6.3 – Résultats en terme de nombre de répliques, de pourcentage de violations, de la consommation énergétique et de la dépense par expérience, pour chaque seuil de bande-passante
Moyenne (écart-type)

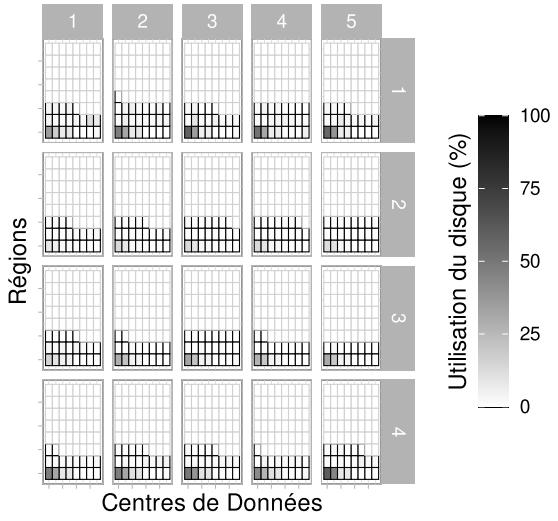
Il est intéressant de comparer ces stratégies du point de vue de l'utilisation du stockage et du placement des répliques. La figure 6.3 représente l'utilisation du disque des nœuds pour les valeurs de seuils *Low*, *Mid* et *High*. Nous voyons que plus la valeur du seuil est élevée plus les répliques seront concentrées sur un faible nombre de nœuds.

Un résumé des résultats est présenté sur le tableau 6.3. Nous voyons ainsi, que plus la valeur du seuil augmente, plus la consommation énergétique diminue, avec une réduction de plus de 20% entre le *Very Low* et le *Very High* sur la seconde expérience. Cela s'explique par le fait que plus le seuil est élevé, plus les répliques seront concentrées sur peu de nœuds permettant d'éteindre les nœuds inactifs. A l'inverse, plus le seuil est faible, plus la stratégie va allumer de nœuds et donc consommer plus d'énergie. De plus, comme nous l'avons présenté précédemment, la stratégie va répliquer une donnée, lorsqu'une tâche nécessitant cette donnée sera exécutée dans le bon centre de données. S'il est nécessaire de transférer la donnée à chaque réplique, cela va augmenter la consommation réseau du centre de données, et donc augmenter le nombre de violations, les coûts liées aux violations et enfin augmenter le nombre de répliques à créer.



(A) Stockage de *Low*

(B) Stockage de *Mid*



(C) Stockage de *High*

FIGURE 6.3 – Utilisation du stockage par nœud pour différentes valeurs de seuil (Low, Mid, High)

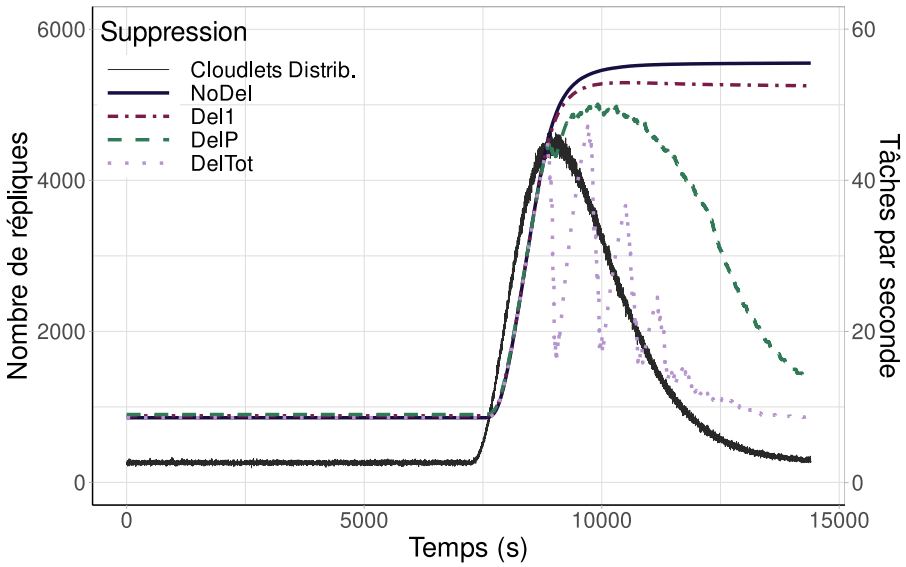


FIGURE 6.4 – Réplication selon le nombre de suppressions - Expérience 1

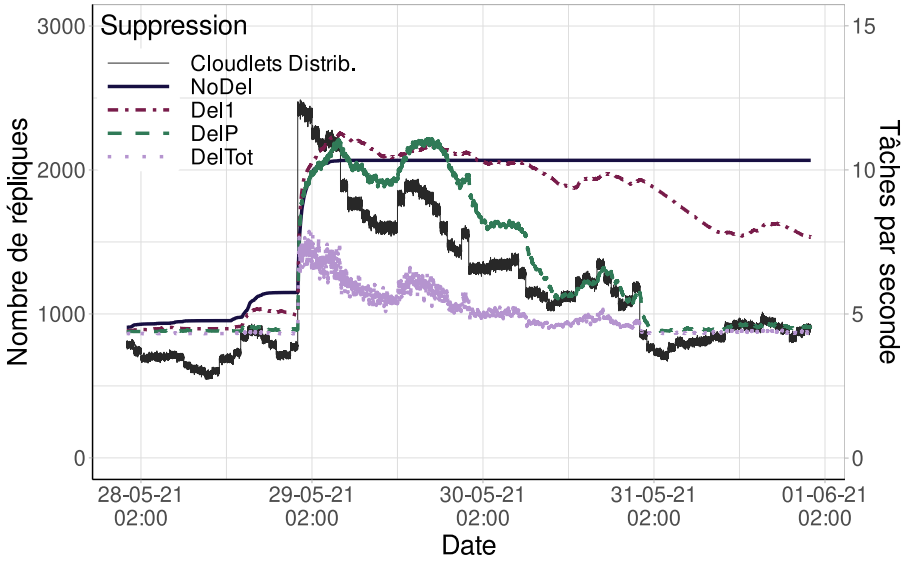


FIGURE 6.5 – Réplication selon le nombre de suppressions - Expérience 2

	Paramètres	Violations	Énergie (MJ)	Dépense (k$)
XP1	NoDel	4.14% (0.27%)	209.7 (3.9)	4.23 (0.23)
	Del1	4.11% (0.21%)	208.2 (3.9)	4.24 (0.16)
	DelP	5.66% (0.28%)	204 (4.2)	5.9 (0.28)
	DelTot	13.75% (0.49%)	212.2 (2.3)	7.38 (0.28)
XP2	NoDel	0.17% (0.02%)	4159 (52.9)	56.59 (4.31)
	Del1	0.54% (0.03%)	3919 (14)	90.38 (4.08)
	DelP	1.95% (0.13%)	3959 (15.7)	145.81 (9.8)
	DelTot	4.28% (0.6%)	3984 (23.9)	162.73 (12)

TABLE 6.4 – Résultats en terme de pourcentage de violations, de consommation énergétique et de dépense par expérience, pour chaque proportion de suppressions
Moyenne (écart-type)

6.5.2 Comparaison du nombre de répliques supprimées

Dans la partie 6.4.1, nous avons proposé de comparer les différentes possibilités de nombre de répliques à supprimer lorsque la carte de contrôle se déclenche suite à un faible nombre de violations. Ainsi, nous comparons la suppression réplique par réplique (*Del1*), puis la suppression de 10% de répliques créées (*DelP*) et enfin toutes les répliques créées (*DelTot*). Ces possibilités de suppressions sont comparées avec l'absence de suppression (*NoDel*).

Il est à noter que les répliques supprimées sont uniquement les répliques créées après le placement initial.

Le tableau 6.4 résume les résultats obtenus pour chaque expérience. Nous voyons sur ce tableau et à l'aide des figures 6.4 et 6.5, que le nombre de répliques diminue avec l'augmentation de la proportion de répliques supprimées et à l'inverse le nombre de violations augmente.

Un résultat intéressant est le fait que plus il y a de suppressions plus la stratégie coûte cher. Cela s'explique par le fait que moins il y a de répliques, et plus il sera nécessaire de transférer les données pour répondre aux requêtes. Ces transferts coûtent plus chers que l'économie réalisée par la réduction de coût de stockage.

Du point de vue énergétique, nous voyons que sur l'expérience de court-terme, la stratégie qui consomme le moins est celle qui supprime un pourcentage de répliques créées (*DelP*), ce qui n'est pas le cas dans l'expérience long-terme. Comme nous le voyons sur la figure 6.4, lorsque le pic de charge commence, l'ensemble des politiques augmentent leur nombre de répliques créées, puis lorsque la charge atteint un plateau, la suppression se met en place. Or, *DelTot* augmente à nouveau le nombre de violations entraînant un nouveau cycle de réplication-suppression augmentant la consommation énergétique. Cette augmentation est de l'ordre de 1.1% par rapport à *NoDel*. Ce schéma n'apparaît pas dans *Del1* et *DelP* qui permettent de réduire la consommation énergétique de respectivement 0.7% et 2.6%.

À l'inverse, dans l'expérience long terme, la charge plus faible implique moins de violations en proportion et donc des alternances réplication-suppression avec moins d'amplitude. Comme nous le voyons sur la figure 6.5, les baisses et augmentations de charge permettent de réduire la consommation énergétique durant les creux, mais à chaque augmentation de charge, de nouvelles répliques sont créées. Nous voyons malgré cela que ces créations de répliques augmentent la consommation énergétique par rapport à la suppression un par un qui nécessite moins de créations de répliques.

Un dernier résultat intéressant est que les cartes de contrôle se basant sur les violations ont permis d'adapter le nombre de répliques à la charge de travail. Nous voyons ainsi, que plus il y a de suppressions, plus le nombre de répliques sera sous-évalué, mais suivra nécessairement les augmentations de charge.

Dans la suite des expériences, nous considérerons le nombre de suppressions associé à *DelP* qui supprime lorsque c'est nécessaire 10% des répliques créées après le placement initial.

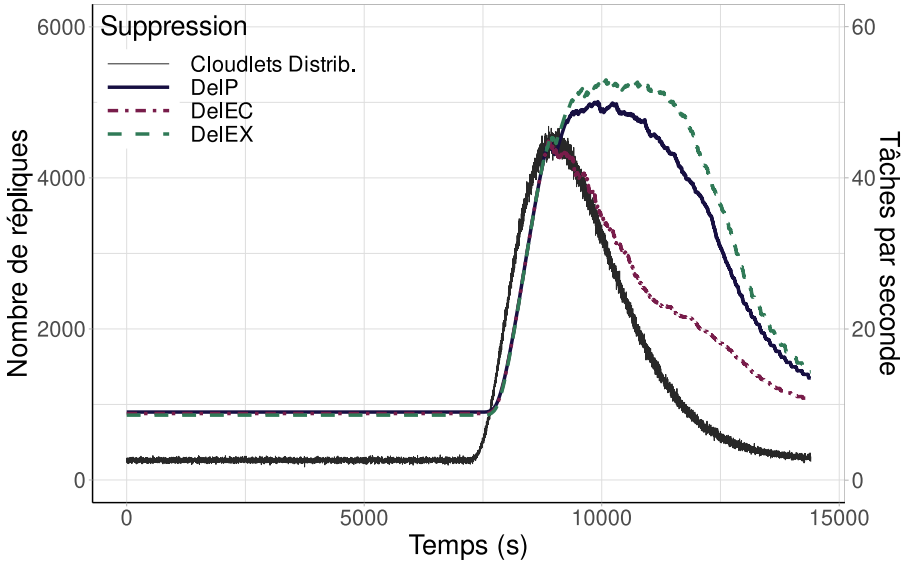


FIGURE 6.6 – Réplication selon la politique de suppression - Expérience 1

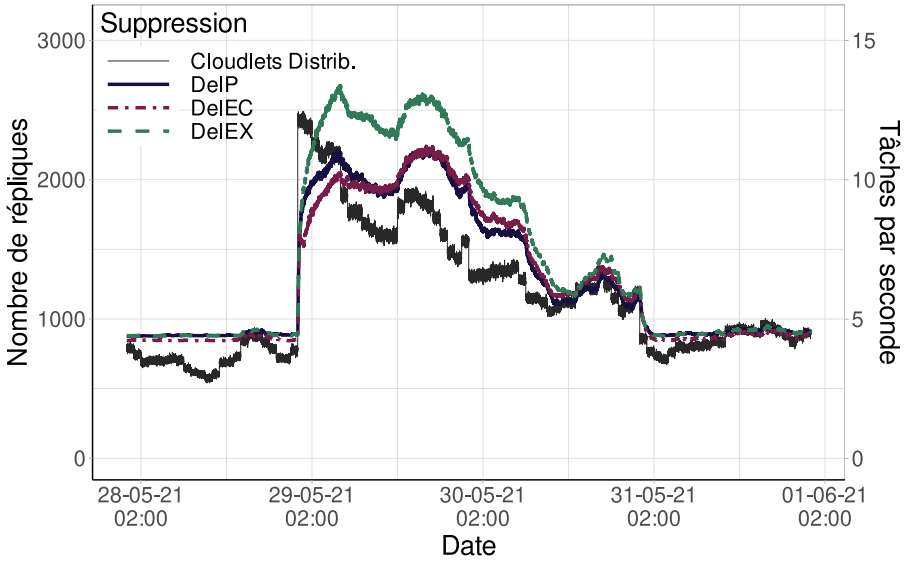


FIGURE 6.7 – Réplication selon la politique de suppression - Expérience 2

	Paramètres	Violations	Énergie (MJ)	Dépense (k$)
XP1	DelP	5.66% (0.28%)	204 (4.16)	5.9 (0.28)
	DelEC	5.55% (0.3%)	185.1 (1.89)	5.76 (0.27)
	DelEX	5.86% (0.39%)	219.2 (5.8)	6.07 (0.28)
XP2	DelP	1.95% (0.132%)	3959 (16)	145.81 (9.8)
	DelEC	2.06% (0.155%)	3977 (21)	148.89 (8.22)
	DelEX	2.17% (0.19%)	4757 (160)	153.96 (12.07)

TABLE 6.5 – Résultats en terme de pourcentage de violations, de consommation énergétique et de dépense par expérience, pour chaque politique de suppression Moyenne (écart-type)

6.5.3 Comparaison entre politiques des scores de suppression

Nous allons à présent comparer l'impact de différents poids (propEC) associés au score. Comme nous l'avons présenté dans la partie 6.4.2, pour choisir la donnée à répliquer, un score a été mis en place tenant compte principalement de la consommation énergétique et de la dépense associées au stockage. La stratégie permet de mettre en avant un de ces objectifs avec une variable qui correspond à la proportion du score de consommation énergétique qui sera pris en compte dans le score final. Par défaut, chacun des scores est centré et réduit (en soustrayant la moyenne et divisant par l'écart-type) permettant d'avoir une même moyenne et variance mettant en relief les répliques qui consomment beaucoup ou qui coûtent particulièrement cher en stockage.

Comme pour les politiques du placement initial, nous allons comparer 3 politiques de suppressions différentes. La première (i) tient compte uniquement de la consommation énergétique (propEC = 1) notée *DelEC*. A l'inverse (ii) une politique qui tient compte uniquement de la dépense (propEC = 0), *DelEX*. Enfin, (iii) une politique intermédiaire qui tient compte des deux objectifs de manière équivalente (propEC = 0.5) : *DelP*. Un résumé des résultats par expérience est présenté sur le tableau 6.5.

Nous voyons ainsi que sur les deux expériences (long terme et court terme), la politique qui prend en compte uniquement la dépense (DelEX) est celle qui augmente à la fois la consommation énergétique et la dépense. Cela s'explique par le fait que supprimer en se basant sur le coût du stockage, va permettre de supprimer les répliques dans les centres de données où c'est le plus cher. Or, supprimer les répliques où le stockage est le plus cher, implique une disparition de ces données dans certains centres de données, augmentant le nombre de transferts, augmentant ainsi la consommation énergétique et la dépense. De plus, comme cela est visible sur le résumé, DelEX augmente le nombre de violations impliquant un maintien élevé du nombre de répliques. Nous voyons ainsi sur les figures 6.6 et 6.7, que DelEX garde le nombre le plus élevé de répliques.

À l'inverse, la prise en compte de la consommation énergétique (DelEC) dépend à la fois des composants du centre de données, mais aussi de la quantité de répliques à supprimer pour pouvoir éteindre un nœud. Si une réplique de petite taille est la seule donnée stockée du nœud, alors la réplique aura un score plus élevé qu'une réplique stockée avec d'autres sur un nœud ayant les mêmes caractéristiques. Les répliques volumineuses sont donc moins fréquemment supprimées et l'extinction d'un nœud est la priorité quelque soit la réplique. Ainsi, moins de suppressions complètes d'une donnée au sein d'un centre de données, avec comme conséquence, moins de transferts, rendant DelEC la politique la moins chère et la moins énergivore dans la première expérience.

En analysant les résultats de DelP, nous constatons que celle-ci consomme en moyenne 7% moins et coûte 3.8% moins que DelEX dans la première expérience. Dans la même expérience, DelP consomme 10% plus que DelEC tout en coûtant 2.4% plus cher. Cependant, dans l'expérience à long terme, les différences entre DelP et DelEC s'amenuisent avec une réduction de moins d'un pourcent pour la consommation énergétique et une différence non significative pour la dépense. A l'inverse, dans cette même expérience, les différences entre DelP et DelEX s'agrandissent avec une réduction de 17% de la consommation énergétique et un réduction de 5.3% de la dépense.

Dans la dernière évaluation, nous gardons donc la politique intermédiaire, qui prend en compte la consommation énergétique et la dépense.

6.5.4 Évaluation Globale de la stratégie dynamique

Dans cette partie, nous allons évaluer la stratégie de réplication dans sa globalité avec le placement initial présenté dans le chapitre 5 et la partie dynamique présentée dans ce chapitre avec la création et la suppression de répliques. Comme présenté dans le déroulé de la stratégie proposée, cette stratégie est nommée *DE2ARS*. Comme pour l'évaluation de la partie statique présentée dans la partie 5.4.4 du chapitre 5, nous comparons ses performances à celles des mêmes stratégies de réplication issues de la littérature.

Ainsi, nous allons nous comparer à une réplication de base nommée **3Rand** qui crée et place 2 répliques de manière aléatoire. Puis, nous allons nous comparer à **PEPR** [Tos et al., 2017] qui

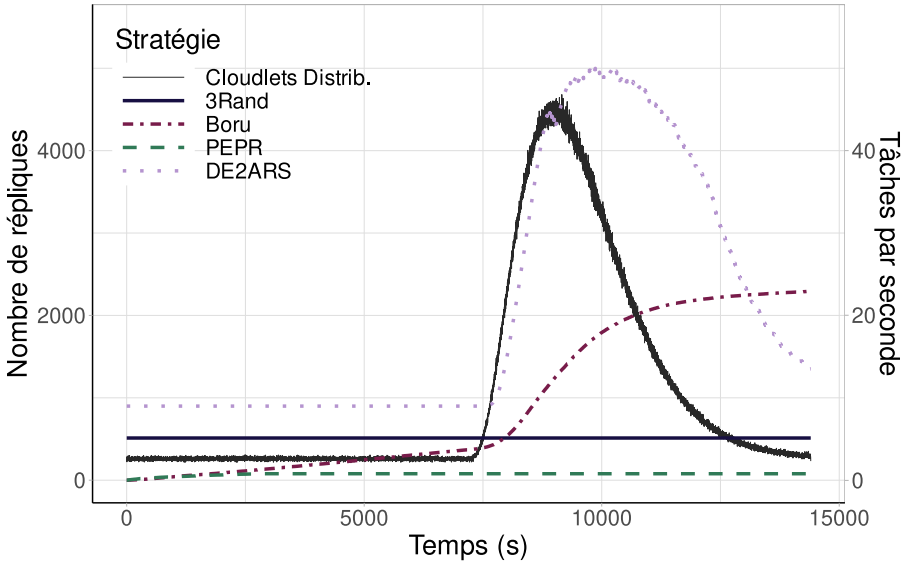


FIGURE 6.8 – Réplication selon la stratégie - Expérience 1

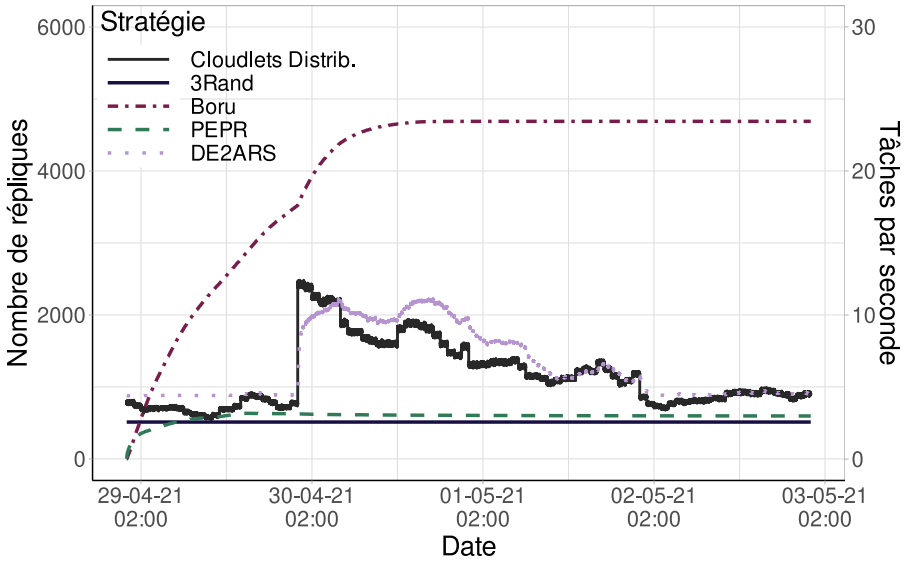


FIGURE 6.9 – Réplication selon la stratégie - Expérience 2

	Paramètres	Violations	Énergie (MJ)	Dépense (k$)
XP1	3Rand	63.23% (1.03%)	594 (22.3)	18.96 (0.62)
	MORM	0% (0%)	501 (18.2)	10.25 (9.68)
	Boru	90.53% (0.42%)	347 (0.3)	4.48 (0.02)
	PEPR	58.92% (3.7%)	552 (59.1)	22.9 (1.08)
	DE2ARS	5.66% (0.28%)	204 (4.2)	5.9 (0.28)
XP2	3Rand	22.97% (1.91%)	7006 (63.8)	259.5 (6.5)
	MORM	0% (0%)	11570 (5)	14.8 (3.1)
	Boru	36.25% (0.6%)	3877 (1.2)	53.2 (0.1)
	PEPR	4.35% (0.57%)	4768 (0.3)	155.6 (5.4)
	DE2ARS	1.95% (0.13%)	3959 (15.7)	145.8 (9.8)

TABLE 6.6 – Résultats en terme de pourcentage de violations, de consommation énergétique et de dépense par expérience, pour chaque stratégie de réplication
Moyenne (écart-type)

s'appuie sur les violations et le profit, ainsi que **MORM** [Long et al., 2014] et **Boru et al.** [Boru et al., 2015] qui considèrent dans leurs objectifs la consommation énergétique et le réseau.

Un résumé des résultats obtenus est présenté sur le tableau 6.6.

Dans un premier temps, il est intéressant d'étudier l'évolution du nombre de répliques dans le temps. Les figures 6.8 et 6.9 représentent le nombre de répliques créées chaque seconde dans la première (court terme) et deuxième (long terme) expérience respectivement. Sur ces résultats n'est pas affichée la stratégie *MORM* qui crée plus de 25 000 répliques dans les deux expériences. Ce nombre n'évolue pas dans le temps, car cette stratégie est statique et les stratégies présentées ayant des pics autour 5000 au maximum, cela aurait rendu les figures moins lisibles.

Vu que les stratégies *PEPR* et *Boru et al.*, n'appliquent pas de placement initial, elles démarrent donc avec aucune réplique. Ainsi, *PEPR* est la stratégie qui réplique le moins malgré un nombre élevé de violations. Cela s'explique par le fait que pour répliquer, *PEPR* se base sur les violations pour déclencher la réplication et applique la réplication si les revenus dépassent la dépense. Lorsqu'il n'y a plus de profit, la stratégie arrête de répliquer. Cela se remarque particulièrement sur la deuxième expérience avec une augmentation progressive au début permettant de répliquer plus de 2 fois, suivit d'aucune réplique créée à la suite des 12 premières heures.

La stratégie *Boru et al.* crée le plus de répliques à la fin des expériences et reste cependant la stratégie qui génère le plus de violations. Comme nous l'avons explicité dans l'évaluation précédente, cela s'explique principalement par l'architecture hiérarchique mise en place, avec une base de données centrale qui génère un goulot d'étranglement et donc plus de violations.

Enfin, la stratégie *DE2ARS* proposée dans ce manuscrit suit la charge de travail à laquelle elle doit s'adapter. De plus, nous voyons que dans les expériences sur lesquelles s'appuient ces résultats, la stratégie proposée est la deuxième stratégie qui a le moins de violations avec un pourcentage de violation de 3.4% pour la première expérience et 2% pour la seconde.

Du point de vue de la consommation énergétique et de la dépense, comme dans le chapitre précédent *MORM* est l'une des stratégies les moins chères avec un coût élevé dû au nombre de répliques créées, mais des coûts faibles sur le long terme, car quasiment aucun transfert n'est nécessaire. Cependant, si la charge devient particulièrement faible, alors les coûts de stockage dépasseront rapidement ce même coût pour les autres stratégies. De même, la consommation énergétique fait partie des plus élevées, car l'ensemble des nœuds restent allumés durant l'expérience.

À l'inverse, *Boru et al.* est l'une des stratégies les moins coûteuses et énergivores. Cela s'explique par l'architecture qui implique qu'un faible nombre de noœuds stockent des répliques, et que les données sont avant tout répliquées dans chaque région puis dans chaque centre de données, permettant de réduire les coûts.

La stratégie *PEPR* est celle qui dépense le plus par rapport aux autres stratégies de la littérature. Cela s'explique principalement par le faible nombre de répliques créées, qui implique une grande quantité de transferts augmentant la dépense.

Enfin, nous voyons que *DE2ARS* permet de réduire la consommation énergétique de 66% et 43% sur les expériences 1 et 2 respectivement par rapport à *3Rand*. De même, la dépense est réduite de 69% pour l'expérience 1 de 44% pour la seconde expérience.

6.6 Analyse des résultats

Dans cette dernière partie, nous ferons une analyse des résultats du point de vue des choix qui ont été fait permettant de mettre en avant d'autres pistes d'améliorations.

6.6.1 Dépense vs. Énergie

Tout au long de la mise en place de la stratégie DE2ARS, nous avons comparé et opposé le choix de tenir compte uniquement de la consommation énergétique et tenir compte uniquement de la dépense. Dans la partie statique, cette opposition est parfois efficace permettant d'un côté de réduire la consommation énergétique et de l'autre la dépense. Cependant, lorsque la charge

devient élevée, cette différence s'amenuise, voire rend plus efficace la politique qui prend en compte uniquement la dépense. Enfin, comme cela a été discuté dans le chapitre sur le placement initial, la majorité de l'impact sur la consommation énergétique provient du nombre de nœuds que l'on gardera allumés dans le placement initial.

Dans ce chapitre, nous avons renouvelé cette opposition selon le poids que l'on donnait au score de suppression. Un des points faibles principaux de ce calcul du score est le fait que l'on tient compte uniquement du coût et de la consommation énergétique du stockage. Or le coût du stockage est bien plus faible que le coût de transfert d'une donnée. Ainsi, en supprimant des répliques pour réduire les coûts de stockage, on augmente les coûts de transfert. À l'inverse, la réduction de la consommation s'appuie principalement sur des technologies d'extinction de nœuds. Moins il y aura de nœuds allumés, moins la consommation énergétique sera élevée.

6.6.2 Suppression de répliques

Pour réduire le nombre de répliques et donc réduire la consommation énergétique et les coûts de stockage, nous avons mis en place une mécanique de suppression de répliques. Ce mécanisme s'appuie sur un calcul de score pour désigner les répliques à supprimer et un nombre de répliques à supprimer. Comme nous l'avons vu précédemment, le score s'appuie sur le stockage uniquement et serait améliorable si on faisait une estimation de l'impact sur le transfert de données.

De plus, nous voyons que le nombre de répliques supprimées a un impact important et diffère selon la charge. Ainsi, il serait intéressant d'adapter plus précisément le nombre de suppressions selon la charge de travail ou selon le nombre de déclenchements de la suppression avec une augmentation progressive.

6.7 Conclusion

Dans ce chapitre, nous avons introduit la partie dynamique de la stratégie de réplication de données. Cette partie dynamique se déclenche à l'aide d'une carte de contrôle, qui est un outil statistique de contrôle qualité. La carte de contrôle s'appuie sur le pourcentage de violations dans chaque échantillon et utilise des règles permettant de générer des alertes. Ces alertes permettent de détecter une augmentation ou une baisse de la charge de travail impliquant une création ou suppression de répliques respectivement. Cette création de répliques se fait en deux étapes avec une étape de planification de la réplication, puis réplication lorsqu'une tâche est exécutée sur le bon nœud avec la bonne donnée. À l'inverse, lorsque le nombre de répliques doit diminuer, on sélectionne un ensemble de répliques à supprimer et elles sont supprimées directement à la suite de la sélection.

L'évaluation des performances nous a permis de constater que la stratégie DE2ARS proposée permet d'adapter le nombre de répliques à la charge de travail. Cela permet de réduire les dépenses de transfert ainsi que le nombre de violations lorsque la charge augmente. Lorsque la charge diminue, la suppression de réplique cherche à la fois à réduire la consommation énergétique, en mettant en veille des nœuds, ainsi que la dépense associée au stockage.

Dans le chapitre suivant, nous allons conclure l'ensemble de ce manuscrit et livrer quelques perspectives possibles.

Chapitre 7

Conclusion et Perspectives

Sommaire

7.1 Synthèse des contributions

Dans ce manuscrit, nous avons proposé une stratégie de réplication de données qui tient compte de la consommation énergétique et de la dépense du fournisseur. Pour répondre à ces problématiques, nous avons modélisé la consommation énergétique et la dépense de différentes actions appliquées aux données, puis nous avons validé ces modèles de consommation énergétique du point de vue des composants. Ces modèles ont été intégrés dans un simulateur pour évaluer l'impact des différentes stratégies comparées du point de vue de la consommation énergétique et de la dépense du fournisseur. À cela s'est ajoutée une implémentation d'une hétérogénéité des nœuds en termes de caractéristiques ainsi que la technique de *Sleep State* permettant de mettre les nœuds en veille. Enfin, pour tirer avantage de cette technique, nous avons implémenté une méthode de placement de tâches naïf qui nécessite la connaissance de l'état du système pour placer la tâche sur un nœud du centre de données.

Dans un premier temps, la stratégie de réplication s'appuie sur un placement initial de répliques, qui vise à répondre à un objectif de disponibilité, en gardant un nombre minimum de répliques, tout en considérant une problématique de performance, en évitant les goulots d'étranglement. Ce placement initial s'appuie un algorithme multi-objectifs afin de minimiser la consommation énergétique et la dépense de ce placement.

L'évaluation des performances liées au placement initial de répliques a permis de mettre en avant l'impact des différents choix de paramètres et de politiques, ayant des impacts fréquemment à la marge. Cependant, par rapport à d'autres stratégies de la littérature, nous arrivons à réduire à la fois la consommation énergétique et la dépense par rapport à la stratégie de base. La stratégie proposée est la stratégie la moins énergivore parmi les stratégies comparées sur l'ensembles des expériences. De plus, ce placement statique a un impact positif sur les performances dans un contexte dynamique.

À ce placement initial s'ajoute une gestion dynamique des répliques. Pour cela, nous nous basons sur les cartes de contrôle qui lèvent des alertes selon qu'elles détectent une augmentation ou baisse de charge de travail. La détection des variations se fait en s'appuyant sur le nombre de violations. Lorsque la carte lève une alerte liée à une augmentation de charge, la stratégie va planifier la création et le placement de nouvelles répliques en s'appuyant sur les violations et la consommation réseau

des nœuds. A l'inverse, s'il y a une baisse de la charge, la stratégie va supprimer des répliques en considérant la consommation énergétique et la dépense du stockage.

L'évaluation de la stratégie proposée dans sa globalité (placement statique et gestion dynamique) met en avant la réduction du nombre de violations liée à cette gestion dynamique. Notamment sur l'importance du choix entre allumer un nouveau nœud pour stocker les répliques et répliquer sur un nœud déjà allumé. De plus, cela a permis de mettre en avant le comportement de la stratégie selon le nombre de répliques à supprimer et le fait de mettre plus en avant la consommation énergétique ou la dépense. Enfin, nous avons pu constater une importante amélioration en terme de performance, avec une légère augmentation de la consommation énergétique tout en réduisant la dépense par rapport au placement statique uniquement.

7.2 Perspectives

Dans cette partie, nous allons discuter de quelques problèmes soulevés lors de la mise en place de la stratégie de réplication proposée, et donc les différentes solutions possibles. Cela nous permettra d'énumérer quelques perspectives possibles à nos travaux, à court, moyen et long terme.

7.2.1 Court terme

Les perspectives à court terme correspondent à de la recherche de l'ordre de quelque mois. Dans un premier temps, nous présenterons les problèmes ouverts durant la mise en place de stratégie de réplication dynamique ainsi que les solutions possibles. Suivi de proposition d'extensions de la stratégie de réplication pouvant être mise en place rapidement.

7.2.1.1 Problèmes ouverts et solutions possibles

Centralisation : La mise en place de la stratégie dynamique, avec la récupération des informations, le déclenchement de la stratégie et la décision de l'ajout et de la suppression de répliques se fait de manière centralisée. Cela implique une augmentation du coût de réseau avec des problématiques de disponibilité d'accès à la stratégie dans un contexte large échelle. Pour répondre à ce problème, il est possible de décentraliser toutes ces étapes dans chaque région, voire chaque centre de données. Dans le cadre d'une décentralisation dans chaque centre de données, une carte est créée pour chaque centre et la décision s'applique au sein de chacun. Cela permet d'uniquement sélectionner les fichiers à répliquer et choisir le nœud sur lequel répliquer.

Surcharge : Il est possible que, dans une architecture large échelle, le nombre de requêtes à prendre en compte puisse devenir particulièrement élevé, augmentant la fréquence d'exécution de la stratégie. Dans la proposition de la stratégie de réplication, la carte considère un ensemble de requêtes pour lever des alertes. Or, les cartes sont prévues pour tenir compte d'un échantillon de produits sur lequel on applique la carte de contrôle. L'idée serait donc d'ajouter une requête ou non à l'échantillon de manière aléatoire. Cela permettrait de contrôler la fréquence d'activation de la carte.

Ces solutions n'ont pas été mises en place durant la thèse, mais seraient des pistes d'amélioration à la suite de la thèse.

7.2.1.2 Extensions possibles

Dans cette partie, nous introduisons les extensions et approfondissement des techniques déjà mises en place.

Une possible extension de la stratégie proposée serait l'ajout d'autres évènements pouvant être pris en compte, en gardant les mêmes mécanismes et méthodes mises en place. Durant la thèse, nous avons considéré trois évènements différents avec (i) un placement initial, (ii) une augmentation et (iii) une baisse de charge. D'autres évènements auraient pu être pris en compte notamment

l'ajout d'un nouveau fichier, avec une réutilisation de l'algorithme de placement initial, et l'ajout ou suppression d'un nœud.

De même, les cartes de contrôle étant déjà mises en place, il y a un approfondissement à faire sur cette technique. Nous avons mis en place une carte de contrôle sur la proportion de violations, permettant de mettre dans une boite noire l'ensemble du processus et de voir l'impact uniquement sur la présence ou non d'une violation de SLO pour une requête. Il serait donc intéressant de voir l'impact des cartes de contrôle à différentes étapes du processus, en plus de l'application de plusieurs cartes dans le processus complet. Cela permettrait de prévenir l'arrivée de violations, en constatant une augmentation progressive du temps de réponse par exemple.

Enfin, une dernière perspective qui pourrait être rapidement mise en place concernerait la manière de valider de la stratégie de réplication de données. En effet, nous avons validé la stratégie de réplication de données au travers de la simulation. Une tâche à court-terme serait d'évaluer la stratégie sur une architecture réelle notamment sur la plateforme Grid5000. Un début de mise en place a été effectuée durant un stage de Master 2 que nous avons encadré Cette implémentation réelle permettrait l'utilisation d'outil de gestion de données plus large échelle sur la plateforme tout en s'appuyant sur un monitoring impliquant différentes métriques telles que la consommation énergétique et la consommation du réseau.

7.2.2 Moyen terme

Les perspectives à moyen terme correspondraient à des travaux qui prendraient entre six mois et un an et demi, correspondant principalement à la prise en compte d'autres techniques d'allocation de tâches et de gestion de données.

Une première extension à étudier à moyen terme, serait d'implémenter différents algorithmes de placement de tâches qui tiennent compte de problématiques énergétiques et économiques. Cela permettrait de s'approcher de la réalité avec des algorithmes issues de la littérature [Guo, 2019, Gu et al., 2020]. De plus, ces algorithmes peuvent s'appuyer sur des techniques que nous n'avions pas mises en place telle que le *Dynamic Voltage and Frequency Scaling* (DVFS) [Snowdon et al., 2005] qui consiste en la réduction de la fréquence du processeur pour en réduire sa consommation énergétique. Ces algorithmes impliqueraient des extensions de la stratégie de réplication pour mieux tenir compte des algorithmes de placement et techniques de réduction de la consommation énergétique.

Une seconde technique en perspective est la technique des codes correcteurs (*Erasure Coding*) [Li and Li, 2013, Burihabwa et al., 2016]. L'idée de cette technique est de construire des blocs à partir d'une donnée. Ces blocs sont construits à l'aide de codes correcteurs. Cela permettrait de construire plus de blocs afin d'atteindre des objectifs de disponibilité. Par exemple, pour répondre à un objectif de 2 pannes simultanées de nœuds de stockage, il y aurait deux solutions. (i) La création de 2 répliques supplémentaires, multipliant par 3 la consommation en terme de stockage. (ii) La division de la donnée en 3 blocs et en créant 2 blocs de codes correcteurs supplémentaires, multipliant la consommation de stockage par $\frac{5}{3}$. Cette technique serait intéressante à intégrer au sein d'une stratégie de réplication, qui appliquerait le choix d'utiliser l'*Erasure Coding* selon la taille de la donnée et sa chaleur.

Enfin, une dernière perspective à moyen terme serait d'améliorer les modèles de consommation énergétique et leurs validations. Une grosse partie de la recherche s'appuie sur la consommation énergétique liée à l'exécution de tâches. Dans notre contexte, nous nous intéressons surtout à la consommation énergétique liée à la gestion de données. Cette extension pourrait passer par l'impact du stockage de données sur un nœud et sur l'interaction entre les composants lors d'un transfert de données par exemple.

7.2.3 Long terme

Les perspectives à long terme seraient des recherches qui durerait plusieurs années. Ces perspectives correspondent à des extensions de la stratégie, et la prise en compte d'autres informations

sur les centres de données. Nous divisons ces extensions long terme en deux axes : environnemental et économique.

Axe environnemental : Une première perspective serait une prise en compte de nouveaux objectifs dans les SLOs pour intégrer les problématiques environnementales, comme l'émission de gaz à effet de serre ou la réduction de la consommation énergétique [Li et al., 2015, Rostirolla, 2019]. Un autre approfondissement intéressant serait la prise en compte de l'émission de gaz à effet de serre. Cette prise en compte, au sein d'une stratégie de réplication de données, peut se voir de différentes manières avec une maîtrise du mélange énergétique pour mettre en avant les énergies faiblement émettrice de gaz à effet de serre lors du placement des répliques [Xu et al., 2015]. À cela peut s'ajouter une considération économique en s'appuyant sur le marché du CO2, en achetant ou revendant des droits d'émission [Zhang and Sun, 2016].

Axe économique : Une perspective intéressante serait la recherche d'information sur l'origine des coûts des fournisseurs et ainsi comment l'utilisation des différences entre pays permet d'optimiser ces coûts [Arvanitis et al., 2017, Cong et al., 2020]. Cela se traduirait par le placement de données dans des centres moins coûteux pour des raisons géographiques, avec une main d'œuvre moins chère par exemple. Une autre prise en compte peut aussi être étudiée du côté du revenu, en tenant compte des comportements utilisateurs et de l'impact d'une violation de niveau de service sur ces derniers. De plus, cela pourrait se coupler avec une mise à jour des objectifs de niveaux de services qui seraient attentifs à des questions environnementales en réduisant les coûts pour les utilisateurs et utilisatrices qui cherchent à réduire leurs émissions de gaz à effet de serre par exemple.

Bibliographie

[iso, 2015] (2015). ISO 9000 :2015(en), Quality management systems — Fundamentals and vocabulary. https://www.iso.org/.

[Aguirre and Tanaka, 2005] Aguirre, H. E. and Tanaka, K. (2005). Selection, Drift, Recombination, and Mutation in Multiobjective Evolutionary Algorithms on Scalable MNK-Landscapes. In Coello Coello, C. A., Hernández Aguirre, A., and Zitzler, E., editors, *Evolutionary Multi-Criterion Optimization*, volume 3410 of *Lecture Notes in Computer Science*, pages 355–369, Berlin, Heidelberg. Springer.

[Alghamdi et al., 2017] Alghamdi, M., Tang, B., and Chen, Y. (2017). Profit-based file replication in data intensive cloud data centers. In *2017 IEEE International Conference on Communications (ICC)*, pages 1–7, Paris, France. IEEE.

[Armbrust et al., 2010] Armbrust, M., Stoica, I., Zaharia, M., Fox, A., Griffith, R., Joseph, A. D., Katz, R., Konwinski, A., Lee, G., Patterson, D., and Rabkin, A. (2010). A view of cloud computing. *Communications of the ACM*, 53(4) :50–58.

[Arvanitis et al., 2017] Arvanitis, S., Kyriakou, N., and Loukis, E. N. (2017). Why do firms adopt cloud computing ? A comparative analysis based on South and North Europe firm data. *Telematics and Informatics*, 34(7) :1322–1332.

[Bai et al., 2013] Bai, X., Jin, H., Liao, X., Shi, X., and Shao, Z. (2013). RTRM : A Response Time-Based Replica Management Strategy for Cloud Storage System. In *Grid and Pervasive Computing*, volume 7861 of *Lecture Notes in Computer Science*, pages 124–133. Springer Berlin Heidelberg.

[Balouek et al., 2013] Balouek, D., Amarie, A. C., Charrier, G., Desprez, F., Jeannot, E., Jeanvoine, E., Lèbre, A., Margery, D., Niclausse, N., Nussbaum, L., Richard, O., Perez, C., Quesnel, F., Rohr, C., and Sarzyniec, L. (2013). Adding Virtualization Capabilities to the Grid'5000 Testbed. In Ivanov, I. I., van Sinderen, M., Leymann, F., and Shan, T., editors, *Cloud Computing and Services Science*, volume 367 of *Communications in Computer and Information Science*, pages 3–20, Cham. Springer International Publishing.

[Belkhir and Elmeligi, 2018] Belkhir, L. and Elmeligi, A. (2018). Assessing ICT global emissions footprint : Trends to 2040 & recommendations. *Journal of Cleaner Production*, 177 :448–463.

[Beloglazov et al., 2012] Beloglazov, A., Abawajy, J., and Buyya, R. (2012). Energy-aware resource allocation heuristics for efficient management of data centers for Cloud computing. *Future Generation Computer Systems*, 28(5) :755–768.

[Bezos, 2017] Bezos, J. (2017). Amazon Annual Report 2016. https://s2.q4cdn.com/299287126/files/doc_financials/annual/Amazon_AR.PDF.

[Bezos, 2020] Bezos, J. (2020). Amazon Annual Report 2020. https://s2.q4cdn.com/299287126/files/doc_financials/2021/ar/Amazon-2020-Annual-Report.pdf.

[Bordage et al., 2021] Bordage, F., De Montenay, L., Vergeynst, O., Montagut, A., and Sauvage, S. (2021). Impacts environnementaux du numérique en France. https://www.greenit.fr/impacts-environnementaux-du-numerique-en-france/.

[Boru et al., 2015] Boru, D., Kliazovich, D., Granelli, F., Bouvry, P., and Zomaya, A. Y. (2015). Energy-efficient data replication in cloud computing datacenters. *Cluster Computing*, 18(1) :385–402.

[Burihabwa et al., 2016] Burihabwa, D., Felber, P., Mercier, H., and Schiavoni, V. (2016). A Performance Evaluation of Erasure Coding Libraries for Cloud-Based Data Stores. In Jelasity, M. and Kalyvianaki, E., editors, *Distributed Applications and Interoperable Systems*, volume 9687 of *Lecture Notes in Computer Science*, pages 160–173, Cham. Springer International Publishing.

[Buyya et al., 2009] Buyya, R., Yeo, C. S., Venugopal, S., Broberg, J., and Brandic, I. (2009). Cloud computing and emerging IT platforms : Vision, hype, and reality for delivering computing as the 5th utility. *Future Generation Computer Systems*, 25(6) :599–616.

[Calheiros et al., 2011] Calheiros, R. N., Ranjan, R., Beloglazov, A., De Rose, C. A. F., and Buyya, R. (2011). CloudSim : a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. *Software : Practice and Experience*, 41(1) :23–50.

[Christensen et al., 2010] Christensen, K., Reviriego, P., Nordman, B., Bennett, M., Mostowfi, M., and Maestro, J. (2010). IEEE 802.3az : the road to energy efficient ethernet. *IEEE Communications Magazine*, 48(11) :50–56.

[Cong et al., 2020] Cong, P., Xu, G., Wei, T., and Li, K. (2020). A Survey of Profit Optimization Techniques for Cloud Providers. *ACM Computing Surveys*, 53(2) :1–35.

[Da Costa, 2021] Da Costa, G. (2021). MojitO/S. https://hal.archives-ouvertes.fr/hal-03453537.

[Da Costa et al., 2020] Da Costa, G., Pierson, J.-M., and Fontoura-Cupertino, L. (2020). Fast maximum coverage of system behavior from a performance and power point of view. *Concurrency and Computation : Practice and Experience*, 32(14) :58–73.

[Dave, 2021] Dave, M. (2021). Seagate 2021 Analyst Day. https://s24.q4cdn.com/101481333/files/doc_downloads/2021/2/2021-Seagate-Analyst-Day.pdf.

[Dayarathna et al., 2016] Dayarathna, M., Wen, Y., and Fan, R. (2016). Data Center Energy Consumption Modeling : A Survey. *IEEE Communications Surveys & Tutorials*, 18(1) :732–794.

[De Courchelle et al., 2019] De Courchelle, I., Guérout, T., Da Costa, G., Monteil, T., and Labit, Y. (2019). Green energy efficient scheduling management. *Simulation Modelling Practice and Theory*, 93 :208–232.

[Deb, 2014] Deb, K. (2014). Multi-objective Optimization. In Burke, E. K. and Kendall, G., editors, *Search Methodologies : Introductory Tutorials in Optimization and Decision Support Techniques*, pages 403–449. Springer US, Boston, MA.

[Ebadi and Navimipour, 2019] Ebadi, Y. and Navimipour, N. J. (2019). An energy-aware method for data replication in the cloud environments using a Tabu search and particle swarm optimization algorithm. *Concurrency and Computation : Practice and Experience*, 31(1) :1–10.

[Fan et al., 2007] Fan, X., Weber, W.-D., and Barroso, L. A. (2007). Power provisioning for a warehouse-sized computer. *ACM SIGARCH Computer Architecture News*, 35(2) :13–23.

[Gill and Singh, 2016] Gill, N. K. and Singh, S. (2016). A dynamic, cost-aware, optimized data replication strategy for heterogeneous cloud data centers. *Future Generation Computer Systems*, 65 :10–32.

[Goyal, 2014] Goyal, S. (2014). Public vs Private vs Hybrid vs Community - Cloud Computing : A Critical Review. *International Journal of Computer Network and Information Security*, 6(3) :20–29.

[Grange, 2019] Grange, L. (2019). *Datacenter management for on-site intermittent and uncertain renewable energy sources*. These de doctorat, Université Toulouse III.

[Grange et al., 2018] Grange, L., Da Costa, G., and Stolf, P. (2018). Green IT scheduling for data center powered with renewable energy. *Future Generation Computer Systems*, 86 :99–120.

[Gu et al., 2020] Gu, C., Li, Z., Huang, H., and Jia, X. (2020). Energy Efficient Scheduling of Servers with Multi-Sleep Modes for Cloud Data Center. *IEEE Transactions on Cloud Computing*, 8(3) :833–846.

[Guo, 2019] Guo, C. (2019). *Allocation de ressources efficace en énergie pour les bases de données dans le cloud*. These de doctorat, Université Toulouse III.

[Hamzaoui et al., 2020] Hamzaoui, I., Duthil, B., Courboulay, V., and Medromi, H. (2020). A Survey on the Current Challenges of Energy-Efficient Cloud Resources Management. *SN Computer Science*, 1(2) :1–28.

[Hazen et al., 2014] Hazen, B. T., Boone, C. A., Ezell, J. D., and Jones-Farmer, L. A. (2014). Data quality for data science, predictive analytics, and big data in supply chain management : An introduction to the problem and suggestions for research and applications. *International Journal of Production Economics*, 154 :72–80.

[Herbst et al., 2013] Herbst, N. R., Kounev, S., and Reussner, R. (2013). Elasticity in Cloud Computing : What It Is, and What It Is Not. In *ICAC*, volume 13, pages 23–27, San Jose, California. USENIX Association.

[Hlavacs et al., 2009] Hlavacs, H., Da Costa, G., and Pierson, J.-M. (2009). Energy Consumption of Residential and Professional Switches. In *2009 International Conference on Computational Science and Engineering*, volume 1, pages 240–246, Vancouver, BC, Canada.

[Hsu and Kshemkalyani, 2019] Hsu, T.-Y. and Kshemkalyani, A. D. (2019). A Proactive, Cost-aware, Optimized Data Replication Strategy in Geo-distributed Cloud Datastores. In *Proceedings of the 12th IEEE/ACM International Conference on Utility and Cloud Computing*, UCC'19, pages 143–153, New York, NY, USA. Association for Computing Machinery.

[Hylick and Sohan, 2009] Hylick, A. and Sohan, R. (2009). A methodology for generating disk drive energy models using performance data. *Energy (Joules)*, 80 :100.

[Janpet and Wen, 2013] Janpet, J. and Wen, Y.-F. (2013). Reliable and Available Data Replication Planning for Cloud Storage. In *2013 IEEE 27th International Conference on Advanced Information Networking and Applications (AINA)*, pages 772–779, Barcelona. IEEE.

[Jones-Farmer et al., 2014] Jones-Farmer, L. A., Ezell, J. D., and Hazen, B. T. (2014). Applying Control Chart Methods to Enhance Data Quality. *Technometrics*, 56(1) :29–41. Publisher : Taylor & Francis _eprint : https ://doi.org/10.1080/00401706.2013.804437.

[Kilcioglu et al., 2017] Kilcioglu, C., Rao, J. M., Kannan, A., and McAfee, R. P. (2017). Usage Patterns and the Economics of the Public Cloud. In *Proceedings of the 26th International Conference on World Wide Web*, WWW '17, pages 83–91, Republic and Canton of Geneva, CHE. International World Wide Web Conferences Steering Committee.

[Kim et al., 2019] Kim, J., Abdella, G. M., Kim, S., Al-Khalifa, K. N., and Hamouda, A. M. (2019). Control charts for variability monitoring in high-dimensional processes. *Computers & Industrial Engineering*, 130 :309–316.

[Koomey, 2011] Koomey, J. (2011). Growth in data center electricity use 2005 to 2010. Technical Report 9, Analytical Press, completed at the request of The New York Times.

[Kumar et al., 2014] Kumar, K. A., Quamar, A., Deshpande, A., and Khuller, S. (2014). SWORD : workload-aware data placement and replica selection for cloud data management systems. *The VLDB Journal*, 23(6) :845–870.

[Lamehamedi et al., 2002] Lamehamedi, H., Szymanski, B., Shentu, Z., and Deelman, E. (2002). Data replication strategies in grid environments. In *Fifth International Conference on Algorithms and Architectures for Parallel Processing, 2002. Proceedings.*, pages 378–383, Beijing, China.

[Larg, 2020] Larg, A. (2020). Seagate Q4 Report 2020. https://s24.q4cdn.com/101481333/files/doc_financials/2020/q4/FQ4'20-Supplemental-Information-FINAL3.pdf.

[Larg, 2021] Larg, A. (2021). Seagate Q4 Report 2021. https://s24.q4cdn.com/101481333/files/doc_financials/2021/q4/STX-Supplemental-FQ4'21-FINAL.pdf.

[Laumanns et al., 2006] Laumanns, M., Thiele, L., and Zitzler, E. (2006). An efficient, adaptive parameter variation scheme for metaheuristics based on the epsilon-constraint method. *European Journal of Operational Research*, 169(3) :932–942.

[Lerman and Ghosh, 2010] Lerman, K. and Ghosh, R. (2010). Information Contagion : an Empirical Study of the Spread of News on Digg and Twitter Social Networks. In *Fourth international AAAI conference on weblogs and social media*, Washington, DC, US. AAAI Publications. arXiv : 1003.2664.

[Li et al., 2019] Li, C., Wang, Y., Chen, Y., and Luo, Y. (2019). Energy-efficient fault-tolerant replica management policy with deadline and budget constraints in edge-cloud environment. *Journal of Network and Computer Applications*, 143 :152–166.

[Li and Li, 2013] Li, J. and Li, B. (2013). Erasure coding for cloud storage systems : A survey. *Tsinghua Science and Technology*, 18(3) :259–272.

[Li et al., 2015] Li, P., Ju, L., Jia, Z., and Sun, Z. (2015). SLA-Aware Energy-Efficient Scheduling Scheme for Hadoop YARN. In *2015 IEEE 17th International Conference on High Performance Computing and Communications, 2015 IEEE 7th International Symposium on Cyberspace Safety and Security, and 2015 IEEE 12th International Conference on Embedded Software and Systems*, pages 623–628, New York, NY, USA. IEEE.

[Li et al., 2011] Li, W., Yang, Y., and Yuan, D. (2011). A Novel Cost-Effective Dynamic Data Replication Strategy for Reliability in Cloud Data Centres. In *2011 IEEE Ninth International Conference on Dependable, Autonomic and Secure Computing*, pages 496–502, Sydney, Australia. IEEE.

[Lin and Shen, 2017] Lin, Y. and Shen, H. (2017). EAFR : An Energy-Efficient Adaptive File Replication System in Data-Intensive Clusters. *IEEE Transactions on Parallel and Distributed Systems*, 28(4) :1017–1030. Conference Name : IEEE Transactions on Parallel and Distributed Systems.

[Liu et al., 2019] Liu, J., Shen, H., Narman, H. S., Lin, Z., and Li, Z. (2019). Popularity-aware Multi-failure Resilient and Cost-effective Replication for High Data Durability in Cloud Storage. *IEEE Transactions on Parallel and Distributed Systems*, 30(10) :2355–2369.

[Liu et al., 2017] Liu, N., Li, Z., Xu, J., Xu, Z., Lin, S., Qiu, Q., Tang, J., and Wang, Y. (2017). A Hierarchical Framework of Cloud Resource Allocation and Power Management Using Deep Reinforcement Learning. In *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*, pages 372–382, Atlanta, GA, USA. ISSN : 1063-6927.

[Lloyd and Rebow, 2018] Lloyd, R. and Rebow, M. (2018). Data Driven Prediction Model (DDPM) for Server Inlet Temperature Prediction in Raised-floor Data Centers. In *2018 17th IEEE Intersociety Conference on Thermal and Thermomechanical Phenomena in Electronic Systems (ITherm)*, pages 716–725, San Diego, CA, USA. ISSN : 2577-0799.

[Long et al., 2014] Long, S.-Q., Zhao, Y.-L., and Chen, W. (2014). MORM : A Multi-objective Optimized Replication Management strategy for cloud storage cluster. *Journal of Systems Architecture*, 60(2) :234–244.

[Malladi et al., 2012] Malladi, K. T., Nothaft, F. A., Periyathambi, K., Lee, B. C., Kozyrakis, C., and Horowitz, M. (2012). Towards energy-proportional datacenter memory with mobile DRAM. In *2012 39th Annual International Symposium on Computer Architecture (ISCA)*, pages 37–48, Portland, OR, USA. ISSN : 1063-6897.

[Mansouri et al., 2017] Mansouri, N., Rafsanjani, M. K., and Javidi, M. M. (2017). DPRS : A dynamic popularity aware replication strategy with parallel download scheme in cloud environments. *Simulation Modelling Practice and Theory*, 77 :177–196.

[Mansouri and Buyya, 2019] Mansouri, Y. and Buyya, R. (2019). Dynamic replication and migration of data objects with hot-spot and cold-spot statuses across storage data centers. *Journal of Parallel and Distributed Computing*, 126 :121–133.

[Masanet et al., 2020] Masanet, E., Shehabi, A., Lei, N., Smith, S., and Koomey, J. (2020). Recalibrating global data center energy-use estimates. *Science*, 367(6481) :984–986. Publisher : American Association for the Advancement of Science.

[Mastelic et al., 2014] Mastelic, T., Oleksiak, A., Claussen, H., Brandic, I., Pierson, J.-M., and Vasilakos, A. V. (2014). Cloud Computing : Survey on Energy Efficiency. *ACM Computing Surveys*, 47(2) :1–36.

[Mauffret et al., 2019] Mauffret, E., Vernier, F., and Monnet, S. (2019). CAnDoR : Consistency Aware Dynamic data Replication. In *2019 IEEE 18th International Symposium on Network Computing and Applications (NCA)*, pages 1–5, Cambridge, MA, USA. ISSN : 2643-7929.

[Meisner et al., 2009] Meisner, D., Gold, B. T., and Wenisch, T. F. (2009). PowerNap : eliminating server idle power. *ACM SIGARCH Computer Architecture News*, 37(1) :205–216.

[Mell and Grance, 2011] Mell, P. and Grance, T. (2011). The NIST Definition of Cloud Computing. http://faculty.winthrop.edu/domanm/csci411/Handouts/NIST.pdf.

[Micron, 2007] Micron (2007). TN-41-01 : Calculating Memory System Power for DDR3. https://www.micron.com/support.

[Micron, 2017] Micron (2017). TN-40-07 : Calculating Memory Power for DDR4 SDRAM. https://www.micron.com/support.

[Mokadem and Hameurlain, 2020] Mokadem, R. and Hameurlain, A. (2020). A data replication strategy with tenant performance and provider economic profit guarantees in Cloud data centers. *Journal of Systems and Software*, 159 :1–18.

[Montgomery and Woodall, 2008] Montgomery, D. C. and Woodall, W. H. (2008). An Overview of Six Sigma. *International Statistical Review / Revue Internationale de Statistique*, 76(3) :329–346. Publisher : [Wiley, International Statistical Institute (ISI)].

[Morley et al., 2018] Morley, J., Widdicks, K., and Hazas, M. (2018). Digitalisation, energy and data demand : The impact of Internet traffic on overall and peak electricity consumption. *Energy Research & Social Science*, 38 :128–137.

[Moyer et al., 2015] Moyer, D., Carson, S. L., Dye, T. K., Carson, R. T., and Goldbaum, D. (2015). Determining the influence of Reddit posts on Wikipedia pageviews. In *Ninth international AAAI conference on web and social media*, pages 75–82, Oxford, UK. AAAI Publications.

[Mubeen et al., 2018] Mubeen, S., Asadollah, S. A., Papadopoulos, A. V., Ashjaei, M., Pei-Breivold, H., and Behnam, M. (2018). Management of Service Level Agreements for Cloud Services in IoT : A Systematic Mapping Study. *IEEE Access*, 6 :30184–30207. Conference Name : IEEE Access.

[Nadella, 2018] Nadella, S. (2018). Microsoft Annual Report 2018. https://www.microsoft.com/investor/reports/ar18/download-center/index.html.

[Nadella, 2020] Nadella, S. (2020). Microsoft 2020 Annual Report. https://www.microsoft.com/investor/reports/ar20/download-center/index.html.

[Nah, 2004] Nah, F. F.-H. (2004). A study on tolerable waiting time : how long are Web users willing to wait ? *Behaviour & Information Technology*, 23(3) :153–163.

[Nelson, 1984] Nelson, L. S. (1984). The Shewhart Control Chart—Tests for Special Causes. *Journal of Quality Technology*, 16(4) :237–239.

[Neubauer, 1997] Neubauer, A. S. (1997). The EWMA control chart : properties and comparison with other quality-control procedures by computer simulation. *Clinical Chemistry*, 43(4) :594–601.

[Orgerie et al., 2014] Orgerie, A.-C., Assuncao, M. D. d., and Lefevre, L. (2014). A Survey on Techniques for Improving the Energy Efficiency of Large-scale Distributed Systems. *ACM Comput. Surv.*, 46(4) :1–31.

[Pamies-Juarez et al., 2011] Pamies-Juarez, L., García-López, P., Sánchez-Artigas, M., and Herrera, B. (2011). Towards the design of optimal data redundancy schemes for heterogeneous cloud storage infrastructures. *Computer Networks*, 55(5) :1100–1113.

[Qu and Xiong, 2012] Qu, Y. and Xiong, N. (2012). RFH : A Resilient, Fault-Tolerant and High-Efficient Replication Algorithm for Distributed Cloud Storage. In *2012 41st International Conference on Parallel Processing*, pages 520–529, Pittsburgh, PA, USA. IEEE.

[Rhu et al., 2013] Rhu, M., Sullivan, M., Leng, J., and Erez, M. (2013). A locality-aware memory hierarchy for energy-efficient GPU architectures. In *2013 46th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 86–98, Davis, California. ACM Press.

[Rostirolla, 2019] Rostirolla, G. (2019). *Ordonnancement dans un centre de calculs alimenté par des sources d'énergie renouvelables sans connexion au réseau avec une charge de travail mixte basée sur des phases*. These de doctorat, Université Toulouse III.

[Roukh et al., 2016] Roukh, A., Bellatreche, L., Tziritas, N., and Ordonez, C. (2016). Energy-Aware Query Processing on a Parallel Database Cluster Node. In Carretero, J., Garcia-Blas, J., Ko, R. K., Mueller, P., and Nakano, K., editors, *Algorithms and Architectures for Parallel Processing*, Lecture Notes in Computer Science, pages 260–269, Grenada, Spain. Springer International Publishing.

[Sato et al., 2005] Sato, H., Aguirre, H. E., and Tanaka, K. (2005). On the locality of dominance and recombination in multiobjective evolutionary algorithms. In *2005 IEEE Congress on Evolutionary Computation*, volume 1, pages 451–458, Edinburgh, UK. ISSN : 1941-0026.

[Sato et al., 2011] Sato, H., Aguirre, H. E., and Tanaka, K. (2011). Genetic Diversity and Effective Crossover in Evolutionary Many-objective Optimization. In Coello, C. A. C., editor, *Learning and Intelligent Optimization*, Lecture Notes in Computer Science, pages 91–105, Berlin, Heidelberg. Springer.

[Schader and Schmid, 1989] Schader, M. and Schmid, F. (1989). Two Rules of Thumb for the Approximation of the Binomial Distribution by the Normal Distribution. *The American Statistician*, 43(1) :23–24. Publisher : Taylor & Francis Group.

[Selvi and Anbuselvi, 2018] Selvi, S. A. E. and Anbuselvi, R. (2018). RAAES : Reliability-Assured and Availability-Enhanced Storage for Cloud Environment. *International Journal of Pure and Applied Mathematics*, 118(9) :103–112.

[Shewhart, 1931] Shewhart, W. A. (1931). *Economic control of quality of manufactured product*. Macmillan And Co Ltd, London.

[Snowdon et al., 2005] Snowdon, D., Ruocco, S., and Heiser, G. (2005). Power Management and Dynamic Voltage Scaling : Myths and Facts. In *20005 Workshop on Power Aware Real-time Computing*, volume 31, page 7, New Jersey, USA.

[Sohan et al., 2010] Sohan, R., Rice, A., Andrew, W. M., and Mansley, K. (2010). Characterizing 10 Gbps network interface energy consumption. In *IEEE Local Computer Network Conference*, pages 268–271, Denver, CO, USA. IEEE.

[Song, 2018] Song, M. (2018). Minimizing Power Consumption in Video Servers by the Combined Use of Solid-State Disks and Multi-Speed Disks. *IEEE Access*, 6 :25737–25746.

[Sousa et al., 2018] Sousa, F. R. C., Moreira, L. O., Costa Filho, J. S., and Machado, J. C. (2018). Predictive elastic replication for multi-tenant databases in the cloud. *Concurrency and Computation : Practice and Experience*, 30(16) :1–15.

[Surbiryala and Rong, 2019] Surbiryala, J. and Rong, C. (2019). Cloud Computing : History and Overview. In *2019 IEEE Cloud Summit*, pages 1–7, Washington, DC, US.

[Tabet et al., 2017] Tabet, K., Mokadem, R., Laouar, M. R., and Eom, S. (2017). Data Replication in Cloud Systems : A Survey. *International Journal of Information Systems and Social Change (IJISSC)*, 8(3) :17–33.

[Tos et al., 2017] Tos, U., Mokadem, R., Hameurlain, A., Ayav, T., and Bora, S. (2017). Ensuring performance and provider profit through data replication in cloud systems. *Cluster Computing*, 21 :1479–1492.

[Valduriez, 1993] Valduriez, P. (1993). Parallel database systems : Open problems and new issues. *Distributed and Parallel Databases*, 1(2) :137–165.

[Vales et al., 2019] Vales, R., Moura, J., and Marinheiro, R. (2019). Energy-aware and adaptive fog storage mechanism with data replication ruled by spatio-temporal content popularity. *Journal of Network and Computer Applications*, 135 :84–96.

[Wang et al., 2011] Wang, S., Liu, J., Chen, J.-J., and Liu, X. (2011). PowerSleep : A Smart Power-Saving Scheme With Sleep for Servers Under Response Time Constraint. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 1(3) :289–298.

[Wei et al., 2010] Wei, Q., Veeravalli, B., Gong, B., Zeng, L., and Feng, D. (2010). CDRM : A Cost-Effective Dynamic Replication Management Scheme for Cloud Storage Cluster. In *2010 IEEE International Conference on Cluster Computing*, pages 188–196, Heraklion, Greece. IEEE.

[Western Electric, 1956] Western Electric, C. (1956). Western Electric - Statistical Quality Control Handbook. http://westernelectric.sectorlink.org/support/statistical-quality-control-handbook.html.

[Xu et al., 2015] Xu, Z., Deng, N., Stewart, C., and Wang, X. (2015). CADRE : Carbon-Aware Data Replication for Geo-Diverse Services. In *2015 IEEE International Conference on Autonomic Computing*, pages 177–186, Grenoble, France. IEEE.

[Yoshida et al., 2015] Yoshida, M., Arase, Y., Tsunoda, T., and Yamamoto, M. (2015). Wikipedia Page View Reflects Web Search Trend. In *Proceedings of the ACM Web Science Conference*, WebSci '15, pages 1–2, New York, NY, USA. Association for Computing Machinery.

[Zhang et al., 2010] Zhang, J., Li, Z., and Wang, Z. (2010). A multivariate control chart for simultaneously monitoring process mean and variability. *Computational Statistics & Data Analysis*, 54(10) :2244–2252.

[Zhang et al., 2015] Zhang, L., Deng, Y., Zhu, W., Zhou, J., and Wang, F. (2015). Skewly replicating hot data to construct a power-efficient storage cluster. *Journal of Network and Computer Applications*, 50 :168–179.

[Zhang and Sun, 2016] Zhang, Y.-J. and Sun, Y.-F. (2016). The dynamic volatility spillover between European carbon trading market and fossil energy market. *Journal of Cleaner Production*, 112 :2654–2663. Publisher : Elsevier.

[Zitzler et al., 2001] Zitzler, E., Laumanns, M., and Thiele, L. (2001). SPEA2 : Improving the strength Pareto evolutionary algorithm. *TIK-report*, 103.