RÉPUBLIQUE ALGÉRIENNE DÉMOCRATIQUE ET POPULAIRE
MINISTERE DE L'ENSEIGNEMENT SUPERIEUR ET DE LA RECHERCHE SCIENTIFIQUE

UNIVERSITÉ LARBI TEBESSI, TEBESSA
Département de Mathématiques et Informatique
Laboratoire de Mathématiques, d'Informatique et Systèmes (LAMIS)

Thèse

Présentée pour l'obtention du diplôme de

DOCTORAT LMD

En

Informatique

Spécialité : **Systèmes d'information coopératifs**

Titre

Contribution pour la tolérance aux fautes dans les systèmes distribués

Présentée et soutenue par

LEDMI Abdeldjalil

Devant le jury :

Dr. AMROUNE Mohamed	MCA	Université Larbi Tebessi, Tebessa	Président
Dr. HIOUAL Ouassila	MCA	Université Abbes Laghrour, Khenchela	Examinateur
Dr. GHERZOULI Mohamed	MCA	Université Constantine 2, Constantine	Examinateur
Dr. BRADJI Louardi	MCA	Université Larbi Tebessi, Tebessa	Examinateur
Pr. BENDJENNA Hakim	Prof	Université Larbi Tebessi, Tebessa	Directeur de thèse
Dr. HEMMAM Soufiane Mounine	MCA	Université Abbes Laghrour, Khenchela	Co-encadreur

Le …/…/…..…

DÉDICACES

Je dédie ce modeste travail,

A Mon très Cher Père

" Je me rappellerai toujours de tous les moments où tu m'as poussé à travailler et à réussir., Cher père j'avoue que si je suis aujourd'hui arrivé à ce niveau et à ce degré de réussite c'est grâce à tes efforts, à ton sens du sacrifice et à tes conseils. J'avoue vraiment et je reconnais que tu étais et demeures pour moi la lumière qui guide mon chemin et qui m'emmène tout droit vers la réussite. Merci et rabbi ichafik nchallah."

"A celle qui était toujours présente pour moi, à tout moment et en toutes circonstances ; celle qui m'a toujours encouragée et soutenu moralement, à ma très chère mère" ;

"A mes très chers frère et sœurs avec qui j'apprends toujours de la vie, des acquis que je n'aurais jamais su avoir sans vous, Saida, Abdellah Makhlouf, Nabil, Toufik, Maissa, et Amina" ;

A la mémoire de mon frère Bousslah :

"Mon frère, comme le temps passe, la douleur en moi reste vivace car rien ne peut combler le vide ainsi que l'infinie tristesse que tu laisses dans notre maison. Allah yarhmak "

"A ma fiancée Sabah " ;

"A mes chères amis Abdelghani, Mohammed, Souhibe, Houssem, Salem, Aimad, Hamza, Salah, Habib … " ;

ABDELDJALIL

Remerciements

Je tiens premièrement à me prosterner en remerciant Allah le tout-puissant de m'avoir donné le courage et la patience afin de terminer ce travail. À l'issue de la rédaction de cette recherche, je suis convaincue que la thèse est loin d'être un travail solitaire. En effet, je n'aurais jamais pu réaliser ce travail doctoral sans le soutien de quelques personnes.

En premier lieu, je tiens à remercier mon directeur de thèse, monsieur **BENDJENNA Hakim** Professeur à l'Université de Tébessa, pour la confiance qu'il m'a accordée en acceptant d'encadrer ce travail doctoral, pour sa grande disponibilité et ses encouragements tout au long de la préparation de cette thèse. Je veux, à cet effet, lui adresser tous mes sincères remerciements pour la confiance accordée ainsi que la grande liberté d'idées et de travail qu'il m'a donné.

Je souhaiterais exprimer ma gratitude à mon Co-encadreur, Monsieur **HEMMAM sofiane Mounine** maître de conférences à l'Université de Khenchela, pour ses multiples conseils et pour toutes les heures qu'il a consacré à diriger cette thèse. J'aimerais également lui dire à quel point j'ai apprécié sa grande disponibilité et son respect sans failles des délais serrés de relecture des documents que je lui ai adressés. Enfin, j'ai été extrêmement sensible à ses qualités humaines d'écoute et de compréhension tout au long de ce travail doctoral.

Je remercie le plus sincèrement possible les membres de jury Dr. AMROUNE Mohamed, Dr. HIOUAL Ouassila, Dr. GHARZOULI Mohamed, Dr. BRADJI Louardi : pour leur attention et l'intérêt porté à cette thèse et qui ont eu l'obligeance d'accepter de juger ce travail.

Comme je présente mes sincères remerciements à tous les membres de LAMIS qu'ils soient professeurs, docteurs et doctorants qui ont contribué à élaborer ce laboratoire de recherche et qui ont su rendre mon travail agréable à travers leurs conseils, leur simple présence ainsi que les différentes journées doctorales et les conférences élaborées par eux.

Résumé

Avec les systèmes de calculs volontaires, les ressources volontaires sont souvent des membres ordinaires ou des personnes en possession de leurs ordinateurs personnels avec accès à une connexion Internet. Même les organisations peuvent également agir entant que volontaires et fournir leurs ressources informatiques. Ces systèmes sont devenus plus attrayants par le fait qu'ils fournissent la puissance de calculs la plus élevée. Cependant, répondre aux besoins des utilisateurs et maintenir les performances du système dans ce type de paradigme est un défi crucial. En conséquence, nous proposons une nouvelle architecture pour les systèmes de calculs volontaires permettant d'équilibrer la charge de manière décentralisée entre les sous-groupes volontaires qui composent le système, et entre les ressources volontaires d'un sous-groupes volontaires de manière centralisée. En outre, notre proposition présente plusieurs avantages : premièrement, la sélection des ressources les plus appropriées en fonction des besoins des usagers et des performances du système. Deuxièmement, estimer la probabilité de défaillance des ressources volontaires en utilisant un modèle de chaîne de Markov à processus stochastique. Les résultats expérimentaux utilisant le simulateur PeerSim sont établis pour vérifier l'efficacité du système proposé. Les résultats obtenus peuvent être considérés comme prometteurs.

Mots-clés :

Les chaîne de Markov discrète, la probabilité de défaillance, l'équilibrage de charge, la redistribution, la tolérance aux pannes, les systèmes de calculs volontaires.

Abstract

With volunteer computing systems, volunteer resources are often represented by ordinary members or people in possession of their personal computer with access to an Internet connection; Even organizations can also act as volunteers and provide their IT resources. These systems have become more attractive because they provide the highest computing power.

However, to satisfy the user requirements and maintaining the system performance in this kind of system is a crucial challenge. As a result, we propose a new architecture for the volunteer computing systems that allow balancing the load between volunteer sub-groups in a decentralized manner, and between resources inside a volunteer sub-groups in a centralized manner. Moreover, our proposal shows more advantages: First, selecting the most appropriate resource according to the user requirements and system performance. Second, estimating the volunteer resource failure probability by using the stochastic process Markov chain model. Experimental results using the PeerSim Simulator is established to verify the efficacy of the proposed system, and promising results are obtained.

Keywords:

Discrete time Markov chains, failure probability, load balancing, redistribution, fault tolerance, volunteer computing systems.

الملخص

مع أنظمة الحوسبة الطوعية ، غالباً ما تكون الموارد الطوعية أعضاء عاديين أو أشخاص يملكون حواسيبهم الشخصية مع إمكانية الاتصال بالإنترنت ؛ حتى المؤسسات يمكنها أيضًا المساهمة كمتطوعين وتوفير موارد تكنولوجيا المعلومات الخاصة بهم. أصبحت هذه الأنظمة أكثر جاذبية لأنها توفر أعلى قوة حوسبة.

ومع ذلك ، يعد تلبية احتياجات المستخدمين والمحافظة على أداء النظام في هذا النوع من النماذج تحديًا حاسمًا. لذلك، نقترح بنية جديدة لأنظمة الحسابات الطوعية لموازنة الحمل بين المجموعات الفرعية التي تشكل النظام بطريقة لامركزية ، وبين الموارد الطوعية من المتطوعين التي تشكل المجموعات الفرعية مركزيا . بالإضافة إلى ذلك ، فإن اقتراحنا يتمتع بمزايا أكثر: أولاً ، اختيار الموارد الأنسب وفقاً لاحتياجات المستخدمين والمحافظة على آداء النظام. ثانيا ، تقدير احتمال فشل الموارد الطوعية باستخدام نموذج سلسلة ماركوف العشوائية. يتم إنشاء النتائج التجريبية باستخدام محاكي PeerSim للتحقق من فعالية النظام المقترح والحصول على نتائج واعدة.

كلمات البحث:

سلاسل ماركوف الزمنية للتنبؤ ، احتمالية الفشل ، موازنة الحمل ، إعادة التوزيع ، التسامح مع الخطأ ، أنظمة الحوسبة التطوعية.

Table des matières

Liste des figures

Liste des tableaux

INTRODUCTION GÉNÉRALE

1. Contexte du travail

Les Desktop Grid et les systèmes de calculs volontaires (DGVCS[1]), également appelés plates-formes de calcul global sont les systèmes d'informatique réparties les plus importants et les plus puissants au monde depuis leur arrivée à la fin des années 90. Ils offrent une abondance de puissance de calcul pour une fraction du coût des supercalculateurs dédiés. Ces systèmes basés sur l'informatique volontaire : les cycles d'inactivité des ordinateurs de bureau et des postes de travail partagés volontairement par les utilisateurs du monde entier via Internet pour calculer les plus petites tâches d'un énorme problème. SETI@home (Anderson, Cobb, Korpela, Lebofsky, & Werthimer, 2002) et Folding@home ("Folding@home,"), sont deux projets bien connus qui exploitent ce type de plates-formes. Ils sont basés sur le logiciel BOINC (Open Infrastructure for Network Computing) de Berkeley (Anderson, 2004). Le premier projet vise à répondre à la question de savoir si nous sommes seuls dans l'univers. Le second facilite la recherche sur les maladies qui simulent le repliement des protéines, la conception informatique de médicaments et d'autres types de dynamique moléculaire.

2. Motivation

Les dernières statistiques de BOINC[2] sont remarquables : 4,49 millions d'utilisateurs, 910,000 de machines et une vitesse de calcul moyenne de 120,1 PetaFLOPS. De nombreuses applications issues d'un large éventail de domaines scientifiques, plus de 30 projets scientifiques - y compris la bio-informatique, la prévision climatique, la physique des particules et l'astronomie - ont utilisé la puissance de calcul offerte par ces systèmes. En général, la planification est gérée dans le modèle maître/travailleur : un serveur distribue des tâches aux clients ou aux travailleurs participants ; ils traitent ensuite les données d'entrée et renvoient les résultats calculés au serveur.

De nombreux chercheurs, tels que des biologistes et des physiciens, utilisent ce type de système pour effectuer de longues opérations et recevoir une réponse dans un court délai.

L'un des aspects cruciaux des plates-formes du calcul globale est l'hétérogénéité et l'extrême volatilité des ressources, leurs propriétaires peuvent les récupérer sans préavis,

[1] DGVCS: Desktop Grids and Volunteer Computing Systems.

[2] Voir les statistiques de BOINC (Berkeley Open Infrastructure for Network Computing), https://boincstats.com/en/stats/projectStatsInfo

ce qui entraîne ce que l'on appelle communément une panne (et diminuer leur degré de disponibilité). En outre, la motivation des utilisateurs à contribuer est multiple, y compris le désir altruiste d'aider ou, plus important encore, l'attribution de points de crédit proportionnellement à la contribution de l'utilisateur.

Avec l'augmentation de nombre des ressources volontaires de ces plate-forme (la puissance de calculs), et le nombre des demandes, beaucoup des travaux se sont intéressés à l'amélioration des performances dans ces systèmes de calculs volontaires plus que répondre aux besoins des usagers. La plupart d'entre eux (Bala & Chana, 2015; Behera & Tripathy, 2014; Ghafarian, Deldari, Javadi, & Buyya, 2013; Hemam & Hioual, 2017; Mokadem, Hameurlain, & Tjoa, 2012; Pitoura, Ntarmos, & Triantafillou, 2012) se sont concentrés sur la récupération des pannes quand l'erreur se produit et non pas à la prédiction des pannes. Dans ce qui suit, on s'intéresse à la prédiction et la sélection des ressources appropriées sous contrainte des usagers dans les systèmes de calculs volontaire.

La plupart des travaux (Behera & Tripathy, 2014; Lázaro, Kondo, & Marquès, 2012; Mokadem et al., 2012; Pitoura et al., 2012; Savio, 2009; Tse, 2013) ne traitent pas le problème de l'équilibrage de charge sous les exigences des utilisateurs. Malheureusement, si les utilisateurs choisissent les ressources les plus efficaces ayant la plus faible probabilité de défaillance, la pression exercée sur ces ressources augmentera. Ce fait provoque l'augmentation du temps de réponse et la surcharge de ces ressources (Anderson et Fedak, 2006). Il est donc difficile de choisir la ressource qui répond aux besoins de l'utilisateur, c'est-à-dire de choisir la ressource la plus efficace (la plus disponible) qui réduit le temps de réponse d'un côté et équilibre la charge entre les ressources composant le système de l'autre côté (garder la performance du système).

3. Contribution

Dans cette thèse, nous proposons une nouvelle architecture basée sur un système de calculs volontaires, où les participants publics fournissent les ressources. Dans cet environnement, nous abordons deux problèmes essentiels : l'équilibrage de la charge entre les ressources et l'optimisation des critères de temps de réponse et de disponibilité. Ces deux critères (le temps de réponse et la disponibilité) sont contradictoires et ont un impact négatif sur la charge entre les ressources. En d'autres termes, si les utilisateurs doivent optimiser le critère de disponibilité, les demandes sont acheminées vers les ressources volontaires les plus importantes disponibles, ce qui provoque une augmentation du temps de réponse sur ces ressources. D'autre part, si les utilisateurs doivent minimiser le temps de réponse, les demandes sont alors acheminées vers les ressources volontaires caractérisées par une faible

disponibilité. Dans ces cas, certaines ressources volontaires sont surchargées et d'autres insuffisamment. Pour régler ce problème, nous proposons une architecture composée de trois composants globaux :

1) Les sous-groupes volontaires contenaient des ressources de volontaires peu fiables.
2) Les utilisateurs qui envoient leurs demandes au switcher balancer en indiquant leurs besoins en matière de préférences.
3) Le système composé d'un ensemble de switchers balancers (super-nœuds) connectés entre eux via une connexion fiable.

Les switchers balancers jouent un rôle important dans ce système, car ils équilibrent la charge entre eux de manière distribuée et chacun équilibre la charge entre ses ressources volontaires, estime la probabilité de défaillance de chaque ressource volontaire afin de l'utiliser ultérieurement, et sélectionne également la ressource volontaire appropriée en fonction des besoins de l'utilisateur (optimisation de temps de réponse et de la disponibilité). Pour ce faire, notre approche est composée de deux étapes.

Lors de la première étape, nous sélectionnons un sous-groupes volontaire actif sous-chargé, puis les ressources volontaires sous-chargées sont sélectionnées pour satisfaire les performances du système. Dans une deuxième étape, nous proposons un algorithme permettant de sélectionner la ressource volontaire approximative considérée comme la meilleure alternative de ressource volontaire en fonction des besoins de l'utilisateur (optimisation du temps de réponse et de la disponibilité). De plus, nous utilisons le modèle de chaîne de Markov à processus stochastique pour estimer la probabilité d'échec de chaque ressource volontaire.

En résumé, cette thèse inclut les contributions suivantes :

1) La proposition d'une architecture basée sur un modèle de switchers/travailleurs incluant un système de quorum.
2) La proposition des deux algorithmes pour équilibrer la charge dans les sous-groupes volontaires extra et intra de manière centralisée et distribuée.
3) La proposition d'un algorithme permettant de satisfaire les besoins de l'utilisateur et les performances du système.
4) Utilisation du modèle de chaîne de Markov à processus stochastiques pour estimer la probabilité d'échec de chaque ressource volontaire.

Publications

Article de journal:

- Ledmi, A., Bendjenna, H., & Mounine, H. S. (2018). Optimizing Both the User Requirements and the Load Balancing in the Volunteer Computing System by using Markov Chain Model. *International Journal of Enterprise Information Sys*tems (IJEIS), *14*(*1*), 35-62.

Articles dans une conférence internationale :

- Ledmi, A., Bendjenna, H., & Hemam, S. M. (2018, October). Fault Tolerance in Distributed Systems: A Survey. In *2018 3rd International Conference on Pattern Analysis and Intelligent Systems (PAIS)* (pp. *1-5*). IEEE.

4. Structure de la thèse

Le manuscrit en plus d'une introduction générale qui introduit la thèse, est organisé comme suit :

- Le premier chapitre (Chapitre 1) met l'accent sur les systèmes distribués, en présentant leurs principales propriétés, fonctionnalités et applications. Ensuite, nous examinerons les différents types de systèmes distribués, ainsi qu'une description détaillée des différentes architectures des réseaux Overlays. Dans la dernière section de ce chapitre, on parlera de l'état de l'art concernant les projets les plus pertinents de la DGVCS[3], classés en fonction de leur contribution. Nous commencerons par le premier projet de recherche documenté sur les DGVCS.
- Le deuxième chapitre (Chapitre 2) : divisé en deux sections, dans la première section nous présentons les concepts de base sur la tolérance aux pannes, et les différentes techniques de tolérance aux pannes dans le paradigme distribué. Dans la deuxième section, nous citerons en détail les différentes techniques de l'équilibrage de charge chacune avec ses avantages. À la fin de ce chapitre, nous citons quelques travaux récents réalisés pour résoudre : le problème d'équilibrage de charge et la tolérance aux pannes dans les différents systèmes distribués.
- Le troisième chapitres (Chapitre 3) : La contribution principale de la thèse est décrite dans ce chapitre. Nous proposons, dans ce chapitre, l'architecture de notre

[3] DGVCS: Desktop Grids and Volunteer Computing Systems.

approche basée sur un système de calculs volontaires. L'architecture proposée doit être structurée de telle sorte que la disponibilité, la transparence, l'équilibrage de charge, la tolérance aux pannes, et le passage à l'échelle soient garantis. Ainsi le mode de fonctionnement de notre système, et nous détaillons les algorithmes d'équilibrage de charge et la tolérance aux pannes proposées.

- Le quatrième chapitre (Chapitre 4 : afin de valider notre approche avec une évaluation expérimentale. Nous avons implémenté nos algorithmes et notre architecture dans un environnement P2P en utilisant PeerSim comme simulateur. Nous avons discuté les résultats obtenus de simulation dans les différents scénarios en fonction de différents paramètres.
- Enfin, nous concluons ce manuscrit en résumant le travail proposé dans notre thèse. Nous récapitulons les défirent problèmes ouverts, et nous décrivons nos perspectives et les travaux futurs.

Remarque : Une description bien détaillée sur le simulateur PeerSim est présentée dans l'Annexe A, en répondant aux questions sur son utilisation, son fonctionnement et l'utilisation des différents protocoles.

Chapitre 1 **LES SYSTEMES DISTRIBUES**

1.1 Introduction

L'informatique distribuée consiste à résoudre des problèmes de calcul à grande échelle en les divisant en plusieurs tâches, chacune exécutée par plusieurs nœuds en réseau. Ces nœuds autonomes formant un *système distribué*. Ce chapitre commence par un aperçu des systèmes distribués, en présentant leurs principales propriétés, fonctionnalités et applications. Ensuite, nous allons nous intéresser aux différents types de systèmes distribués, ainsi qu'une description bien détaillée des différentes architectures des réseaux Overlays.

1.2 Notion sur les systèmes distribués

Diverses définitions des systèmes distribués ont été données dans la littérature. Selon (Tanenbaum & Steen, 2017): « *Un système distribué est un ensemble d'éléments informatiques (ordinateurs ou processus) indépendants qui apparaissent à un utilisateur comme un seul système cohérent*».

Selon cette définition, on peut distinguer deux principales caractéristiques de systèmes distribués. Le premier est qu'un système distribué représente une collection d'éléments informatiques pouvant chacun se comporter indépendamment les uns des autres. Un élément informatique, que nous désignerons généralement sous le nom de *nœud*, peut être un périphérique matériel ou un processus logiciel. Une deuxième caractéristique est que les utilisateurs (qu'il s'agisse de personnes ou d'applications) pensent avoir affaire à un seul système. Cela signifie que d'une manière ou d'une autre, les nœuds autonomes doivent collaborer. La question de savoir comment établir cette collaboration est au cœur du développement de systèmes distribués.

Du fait que l'ensemble des éléments informatiques forment un système entier, la défaillance d'un de ces éléments peut avoir un impact négatif sur le fonctionnement du système et par conséquent introduire des incohérences. La définition de (Lamport, 1987) explique bien cet aspect : «*Un système qui vous empêche de travailler si une machine dont vous n'avez jamais entendu parler tombe en panne*».

D'une autre vue, le calcul distribué peut également être défini comme un domaine de l'informatique qui fait les études des systèmes distribués. Les principaux objectifs de

conception de ces systèmes peuvent être regroupés en deux domaines : High Performance Computing (HPC) et High Throughput Computing (HTC).

- HPC se réfère à l'accent mis sur la performance de la vitesse brute mesurée en FLOPS[4], associée à des calculs scientifiques et d'ingénierie à grande échelle (Hwang, Dongarra, & Fox, 2013).
- Dans le cas de HTC, l'objectif de performance est déplacé vers un haut débit ou le nombre de tâches réalisées par unité de temps, ce qui est caractéristique pour les applications de services Web (Hwang et al., 2013).

On peut dire que les quatre objectifs principaux (Andrew S. Tanenbaum & Steen, 2017) qui devraient être atteints pour que la construction d'un système distribué en mérite l'attention sont :

- Un système distribué devrait rendre les ressources facilement accessibles ***(Supporting resource sharing)***: Un objectif important d'un système distribué est de permettre aux utilisateurs (et aux applications) d'accéder facilement aux ressources distantes et de les partager. Les ressources peuvent être pratiquement de plusieurs natures (les périphériques, les supports de stockage, les données, les fichiers, et *les services).*
- Il devrait cacher le fait que les ressources sont distribuées à travers un réseau ***(Making distribution transparent)***: un objectif important d'un système distribué est de cacher le fait que ses processus et ressources sont physiquement distribués sur plusieurs ordinateurs, éventuellement séparés par de grandes distances. En d'autres termes, il essaie de rendre la distribution des processus et des ressources transparente et invisible, aux utilisateurs finaux et aux applications.
- Cela devrait être ouvert ***(Being open)*** : un autre objectif important des systèmes distribués est d'être ouvert. Un système distribué ouvert est essentiellement un système qui offre des composants qui peuvent facilement être utilisés, ou intégrés dans d'autres systèmes.
- Il devrait être évolutif ***(Being scalable)*** : La connectivité mondiale via Internet, la possibilité d'avoir relativement des puissants ordinateurs de bureau pour les applications de bureau et de stockage. Dans cet esprit, l'évolutivité est devenue

[4] FLOPS — **FL**oating point **O**perations **P**er **S**econd

l'un des objectifs de conception les plus importants pour les développeurs de systèmes distribués.

1.3 Types de systèmes distribués

On se base sur les domaines d'application, (Andrew S. Tanenbaum & Steen, 2017) a fait une distinction entre trois grandes catégories de systèmes de calcul distribués *(Distributed Computing Systems)*, les systèmes d'information distribués *(Distributed Information Systems)*, et les systèmes pervasifs distribués *(Distributed Pervasive Systems)*. Tous ces systèmes sont naturellement distribués.

Dans notre thèse, on se concentre sur les de systèmes de calcul distribués. Une classe importante de systèmes distribués est celle utilisée pour les tâches calculables de haute performance. En effet, on peut distinguer plusieurs sous-groupes dans cette classe :

a) Les clusters

(Ou superordinateurs) qui ont été initialement créés pour fournir une grande collection de ressources de calculs, de stockage et de réseau pour des applications scientifiques et d'ingénierie. Une construction typique d'un tel système consiste en un logiciel et un matériel homogènes (Coulouris, 2011; Hwang et al., 2013). Les nœuds autonomes et dédiés formant un cluster sont situés à proximité les uns des autres (Coulouris, 2011). Les nœuds sont interconnectés dans un réseau à haut débit fiable (Gigabit Ethernet, LAN[5], etc.) (Hwang et al., 2013). Cette conception est utilisée pour fournir un environnement HPC unique et intégré avec une gestion et sous un contrôle centralisés (Coulouris, 2011; Hwang et al., 2013).

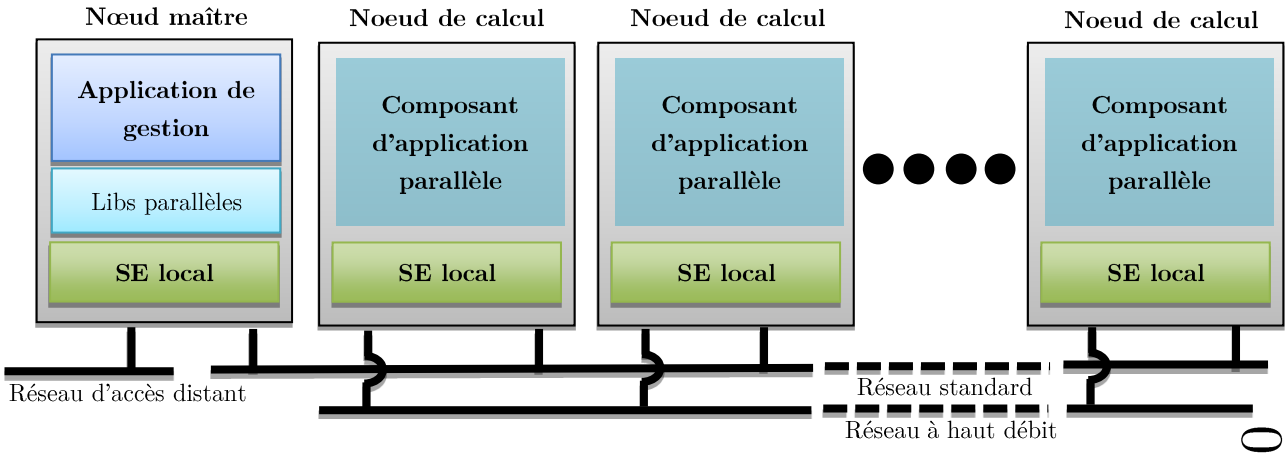

Figure 1.1 Un exemple de système informatique en cluster.

[5] LAN: Local Area Network.

Comme le montre la Figure 1.1, le cluster est utilisé pour la programmation parallèle dans laquelle un seul programme est exécuté en parallèle sur plusieurs nœuds.

b) La grille informatique (Grid)

Avec l'augmentation des besoins en ressources de calculs et de stockage, une idée d'une grille est apparue. Un tel système consiste en grappes faiblement interconnectées à partir de sites géographiquement dispersés (Hwang et al., 2013). Généralement, il s'agit d'une plate-forme hétérogène créée pour assembler et partager des ressources. Une grille est gérée par une organisation multi-institutionnelle (Sheikhalishahi, Devare, Grandinetti, & Incutti, 2012). Malgré tout, chaque propriétaire de cluster conserve le contrôle sur les ressources possédées, ce qui se reflète dans différentes politiques locales et choix de logiciels (Sheikhalishahi et al., 2012). En raison de ce modèle de gestion, une grille est parfois appelée une fédération de clusters.

De nombreux réseaux nationaux et internationaux sont construits dans diverses parties du monde avec le soutien et le financement des gouvernements (principalement) et de l'industrie. ("Grid'5000,") - initialement une plate-forme française couvrant neuf sites, établis à des fins de recherche. Au moment de la rédaction, le projet s'étend (11 sites) en direction internationale et l'université du Luxembourg en est un membre actif. De même, le succès de TeraGrid aux États-Unis (11 sites, terminés en 2011) se poursuit dans une nouvelle entreprise soutenue par 17 institutions - Extreme Science and Engineering Discovery Environment ("XSEDE,").

c) Les systèmes pair-à-pair (peer-to-peer (P2P))

C'est un ensemble d'ordinateurs appelés pairs, qui s'accordent à exécuter une tâche particulière. Les ressources peuvent être des ordinateurs ou des assistants personnels interconnectées via Internet, ce qui rend le système beaucoup plus flexible mais aussi moins fiable et stable.

d) Les Desktop Grids et les Systèmes de calculs volontaires

Actuellement sur les systèmes distribués (Coulouris, 2011), les DGVCS ne sont pas présentés dans une catégorie séparée, mais en combinaison avec des grilles. Cependant, il existe de nombreuses différences entre ces schémas (disponibilité, niveau de sécurité, classe de matériel, etc.), ce qui facilite la distinction.

Généralement, une grille de bureau est composée d'ordinateurs hétérogènes, économiques et partiellement disponibles (PC, postes de travail, etc.) (Barrera, Rosero, & Cano, 2012a).

Un exemple est illustré à la Figure 1.2. Selon la dispersion géographique des machines, le système peut être local ou global. L'utilisation de ressources partagées par un ordinateur (connecté à la grille du bureau) peut être conditionnée par son état, par ex. une exécution de tâche est autorisée uniquement lorsque la machine est inactive ou que l'utilisation d'une ressource est inférieure à un certain niveau. Enfin, le système peut être *privé* ou *public*. Les grilles de bureau privées sont exploitées au sein d'une organisation (comme une entreprise, un laboratoire, une université, etc.), tandis que les grilles publiques reposent principalement sur le bénévolat (c'est-à-dire les contributions de l'utilisateur) ressources.

Afin d'attirer des volontaires, les propriétaires de systèmes proposent diverses incitations basées sur les contributions, telles que les récompenses, les points de crédit, le Temple de la renommée, etc. exécuté dans un système informatique bénévole (global, grille de bureau public).

Les DGVCS sont principalement utilisés pour soutenir des projets scientifiques à grande échelle. Le principal avantage de ces systèmes est la réduction des coûts associés à l'achat et à la maintenance de matériel et d'espaces physiques dédiés (Barrera et al., 2012a). Par exemple, une grille de bureau privée peut être créée dans une organisation donnée en utilisant des ordinateurs personnels, des ordinateurs portables et des postes de travail. Bien entendu, une simple interconnexion de ces ordinateurs ne suffit pas et des moyens supplémentaires sont nécessaires. Condor est le logiciel middleware le plus utilisé pour construire de telles plates-formes. (Renommé HTCondor en 2012 ("Htcondor,")), qui est un système spécialisé de gestion de la charge de travail pour le traitement par lots.

Contrairement aux grilles basées sur des clusters fournissant des environnements HPC, les solutions de bureau offrent des plates-formes HTC requises dans les applications axées sur le traitement d'énormes volumes de données. Les projets informatiques volontaires SETI@home (Anderson et al., 2002) et Folding@home ("Folding@home,") sont bien connus, largement couverts par des exemples médiatiques. D'abord, l'un est utilisé pour rechercher l'intelligence extra-terrestre, le second pour la recherche sur les maladies qui simulent le repliement des protéines, la conception de médicaments et d'autres types de dynamique moléculaire. Grâce à la publicité, ils ont le plus grand nombre de bénévoles. Néanmoins, les projets gourmands en données ne sont pas le seul domaine d'application des grilles de bureau. GIMPS (Giacobini, Tomassini, & Tettamanzi, 2005) (Great Internet Mersenne Prime Search) en est un exemple, car il nécessite principalement une puissance de processeur élevée. Les trois exemples (et beaucoup d'autres) ont une chose en commun : l'utilisation de matériel dédié ne permet pas de réaliser des économies via l'échelle et les ressources nécessaires (Barrera, Rosero, & Cano, 2012b). Cela ne signifie pas que le coût

disparaît lorsque l'utilisation de DGVCS est utilisée, car elle est simplement transmise aux utilisateurs ou aux organisations donatrices. Dans cette thèse, nous nous concentrerons sur les **DGVCS**, dans la section 1.5 nous les détaillerons bien.

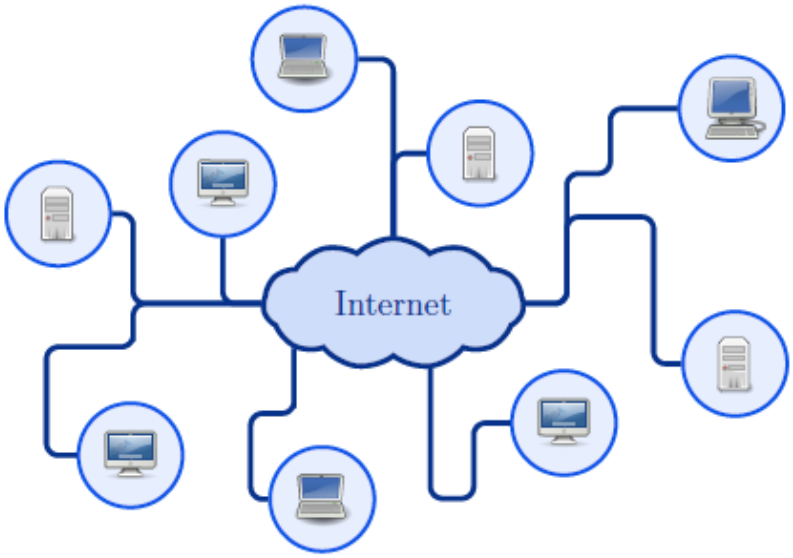

Figure 1.2 Exemple d'un DGVCS.

e) *Cloud* (l'informatique en nuage)

Du point de vue système, le cloud est un ensemble d'ordinateurs (ressources) stockés sur de vastes grilles de serveurs ou de data centres. Les ressources du cloud sont mutualisées à travers une virtualisation qui favorise la montée en charge, la haute disponibilité et un plan de reprise à moindre coût.

Les systèmes d'informatique en nuage *Cloud Computing* (CC) issus du secteur commercial sont axés sur les applications d'entreprise, où les consommateurs utilisent et paient pour ce dont ils ont besoin sur Internet (Sheikhalishahi et al., 2012). Essentiellement, ils mettent en œuvre la vision de l'informatique en tant qu'utilité en proposant « tout en tant que service » (Coulouris, 2011). Les nuages dépendent fortement de ressources évolutives de manière dynamique et souvent virtualisée, qui fournissent un environnement à la demande, instantanée et élastique à la taille et à la fiabilité d'un centre de données. Du point de vue matériel, les nuages sont construits sur des grilles basées sur des clusters. Par conséquent, ces systèmes sont souvent confondus avec les grilles elles-mêmes. Cependant, il existe une différence majeure entre eux : dans les nuages, une ressource est louée à un consommateur qui en a le plein contrôle. Ceci est pris en charge par un hyperviseur sous-jacent, isolant les autres ressources de manière sécurisée. Il est difficile de répondre à la question de savoir si les nuages peuvent fournir suffisamment de performances et de rapidité en matière de

calcul, de stockage et de mise en réseau pour les applications HPC. Elle dépend fortement de la réalisation concrète du nuage (Varrette, Plugaru, Guzek, Besseron, & Bouvry, 2014).

Les services en nuage proviennent des principaux éléments de la technologie de l'information le logiciel s'exécute sur une plate-forme utilisant une infrastructure. Cela se reflète dans différents modèles de services en couches, à la demande, tels que l'**IaaS**, le **PaaS** et le **SaaS**. Ces couches sont illustrées à la Figure 1.3 et sont bien détaillées ci-dessous.

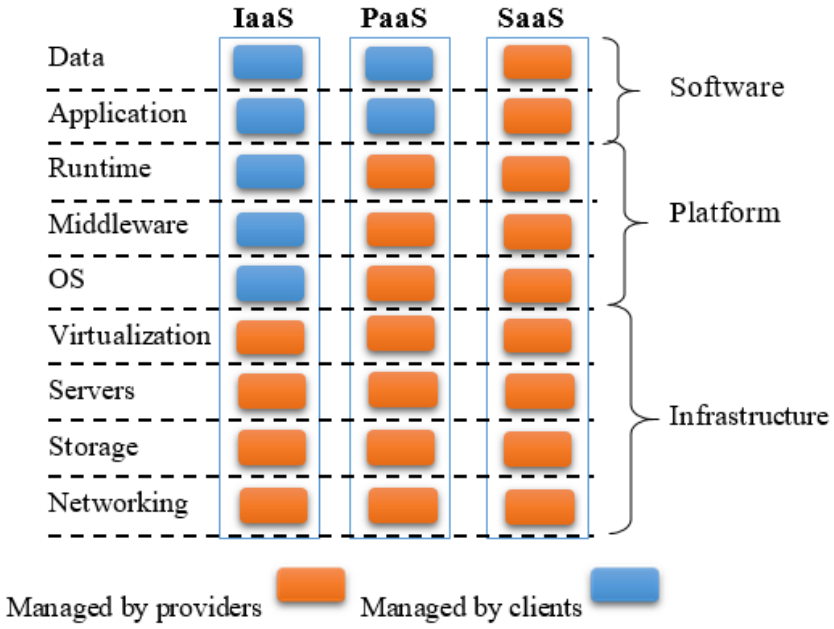

Figure 1.3 Les modèles de services cloud : IaaS, PaaS et SaaS.

(a) Infrastructure en tant que service : Infrastructure as a Service (IaaS)

La couche inférieure (IaaS) repose sur différentes technologies de virtualisation (VT) implémentées sur du matériel physique. Son rôle est de mettre en place des infrastructures composées des ressources informatiques, de stockage et de réseau demandés par les utilisateurs (Hwang et al., 2013). Ceci est réalisé par le déploiement et l'exécution de plusieurs machines virtuelles (VM), exécutant des systèmes d'exploitation invités, généralement sélectionnés par les clients. Les utilisateurs ne contrôlent ni ne gèrent l'infrastructure de cloud sous-jacente (Hwang et al., 2013). Au lieu de cela, ils peuvent superviser le système d'exploitation, le stockage et les applications déployées (Hwang et al., 2013). Parfois, les clients contrôlent certains composants réseau (Hwang et al., 2013). Il existe de nombreux fournisseurs IaaS commerciaux, par exemple: Amazon EC2

("Amazon. Amazon Elastic Compute Cloud (Amazon EC2). ,"), GoGrid ("GoGrid,"), Rackspace ("Rackspace managed cloud.,"). Il est également possible de créer des clouds privés en utilisant l'un des nombreux toolkits disponibles tels que Nimbus ("Nimbus,"), OpenNebula ("OpenNebula,"), Cumulus ("Cumulus Project,"), EUCALYPTUS ("Eucalyptus,"), etc.

(b) Plate-forme en tant que service : Platform as a Service (PaaS)

PaaS est la couche intermédiaire des services de cloud computing. Il permet le développement, le déploiement et la gestion d'applications de génération utilisateur à l'aide d'une plate-forme virtualisée configurée, et bien définie (Hwang et al., 2013). Le PaaS étant orienté sur le cycle de vie des applications, les clients utilisent des infrastructures prédéfinies (Hwang et al., 2013). Cela inclut également les outils logiciels, les langages de programmation et les bibliothèques disponibles. De tels environnements diffèrent généralement entre les fournisseurs. Par exemple, Google App Engine ("Google. Google App Engine: Platform as a Service — Google Cloud Platform. ,") permets aux clients d'exécuter des programmes dans une version limitée de Python ou Java avec un accès à la base de données de Google; Forceforce.com de Salesforce.com ("Salesforce.com Inc. Salesfoce Platform,") prend en charge le développement dans un langage de programmation de type Java, Apex; et Microsoft Azure ("Microsoft. Microsfot Azure,") est orienté .NET.

(c) Logiciel en tant que service : Software as a Service (SaaS)

Enfin, la couche supérieure SaaS fait référence à l'utilisation de logiciels d'application desservant des milliers de clients (Hwang et al., 2013), basée sur un navigateur. Les plates-formes et infrastructures sous-jacentes sont masquées à l'utilisateur. Il existe un vaste choix de services de ce type pour les entreprises et les particuliers: Google Gmail ("Google Gmail,") - gestion des contacts et du courrier électronique, Google Docs - outils bureautiques, gestion de la relation client (CRM) de Salesforce.com ("Salesfoce Platform,"), Facebook ("Facebook,") et beaucoup plus.

1.4 Les différentes architectures des réseaux Overlays

Dans cette section, nous allons décrire les différents concepts de base des réseaux overlay. Nous y fournissons quelques définitions, les motivations d'utilisation de ces systèmes et leurs principales caractéristiques. Ensuite, nous allons entamer les différents modèles de réseaux P2P, ainsi que les avantages et les inconvénients de chaque type.

Les réseaux informatiques sont constitués de divers nœuds et liens de réseau. La façon dont ces périphériques sont connectés définit une topologie. En plus de cela, un réseau overlay peut être utilisé pour spécifier des modèles de communication dans le réseau informatique. Une taxonomie des approches possibles est proposée dans la Figure 1.4. Comme le montre la figure, il existe deux classes principales de réseaux overlay :

1. *Centralisé* lorsqu'un nœud central est connecté aux autres - voir la section 1.4.1.

2. *Peer-to-peer (P2P)* - voir la section 1.4.2.

Les sections suivantes offrent une présentation plus détaillée de ces classes et de leurs ramifications respectives. Des exemples choisis des différentes catégories sont également illustrés à la Figure 1.4.

1.4.1 **Modèles centralisés**

Les réseaux overlay centralisés ont une structure simple : un nœud (ayant un rôle central dans le système) est connecté aux autres. La communication dans une telle organisation suit un modèle de messagerie demande-réponse. L'initiation de l'échange de messages est généralement restreinte, le nœud central envoie des demandes ou leur répond. Par conséquent, il existe deux modèles de réseaux overlay centralisés : *client-serveur* et *maître/travailleur*. La plupart des applications trouvées sur Internet suivent l'architecture client-serveur. Dans ce contexte, un serveur fournit des ressources ou des services pouvant être demandés par un ou plusieurs clients. Le serveur attend les demandes entrantes. Chaque demande reçue est traitée et répond au client.

Dans le modèle maître/travailleur, cette relation est inversée. Un nœud maître envoie des demandes de ressources ou de services aux nœuds de travail. Chaque travailleur attend les demandes entrantes. Chaque demande reçue est traitée et renvoyée au nœud maître.

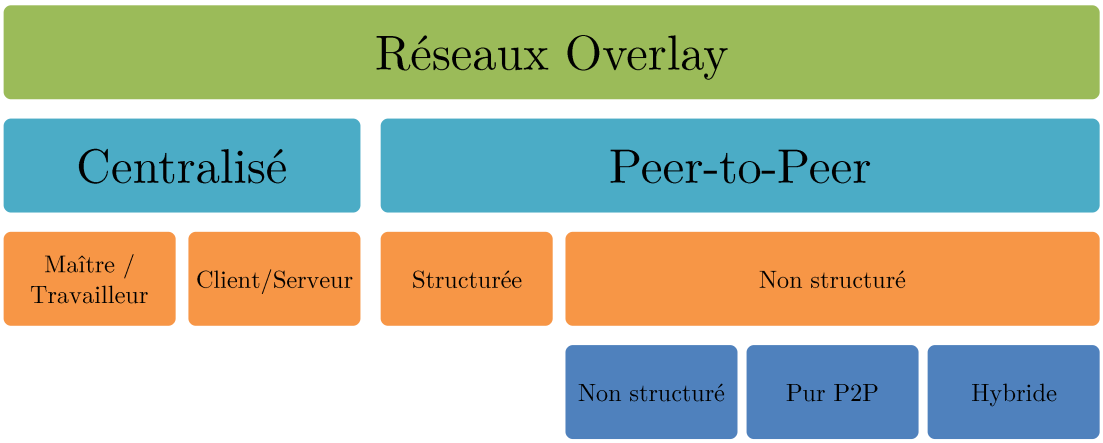

Figure 1.4 Taxonomie des réseaux overlay.

Il n'est pas difficile d'imaginer que le nœud central forme un goulot d'étranglement (en particulier dans l'architecture client-serveur). Par conséquent, un périphérique ayant ce rôle doit être soigneusement sélectionné. Sa configuration (puissance de calcul, mémoire, stockage et bande passante réseau) doit correspondre aux exigences de la charge de travail attendue. Dans certains cas, des systèmes d'équilibrage de charge et de basculement sont introduits pour améliorer l'évolutivité de la solution.

1.4.2 **Modèles Peer-to-peer**

Les réseaux de recouvrement peer-to-peer (P2P) ont été créés pour résoudre les problèmes d'évolutivité des applications distribuées volumineuses. Contrairement aux modèles décrits précédemment, les réseaux présentés ici sont généralement constitués de participants tout

aussi privilégiés et équipotents, appelés pairs. Dans ces schémas, la symétrie dans les rôles est présente, un client peut également être un serveur.

La vue globale est la liste de tous les membres du réseau. Chaque participant conserve une liste de quelques autres pairs représentant sa connaissance (partielle) de l'effectif global, appelée vue partielle. La maintenance de la vue globale par chaque pair est irréaliste dans un système dynamique à grande échelle, car elle entraîne des coûts de synchronisation considérables pour chaque nœud entrant ou sortant (Jelasity, Voulgaris, Guerraoui, Kermarrec, & Van Steen, 2007). Il existe deux sous-types principaux de réseaux overlay

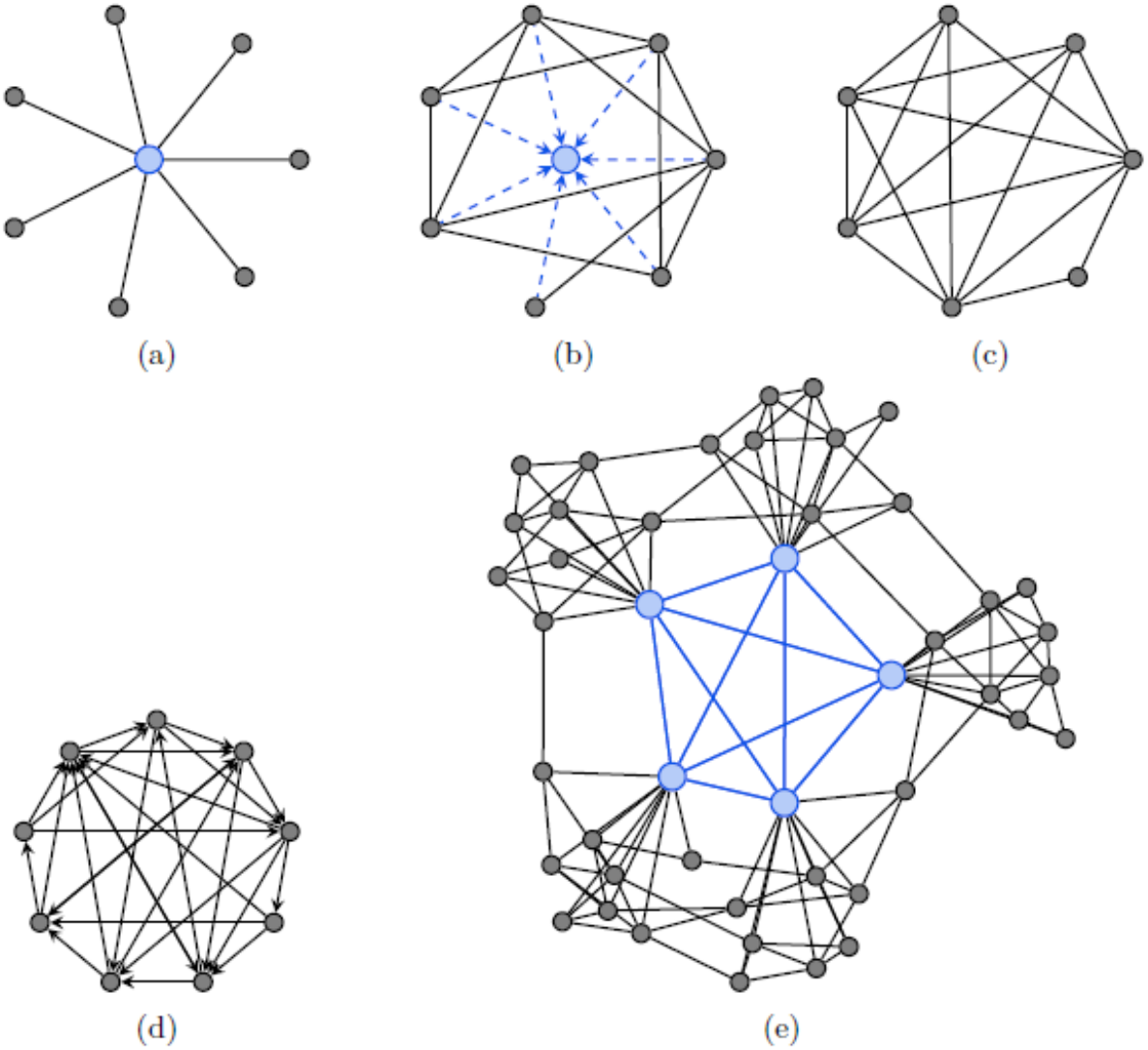

Figure 1.5 Exemples de différents réseaux overlay : (a) centralisé - client-serveur ou maître/travailleur ; (b) P2P centralisé non structuré ; (c) P2P pur non structuré ; (d) P2P structuré (Chord) ; (e) P2P hybride non structuré.

décentralisés : structurés et non structurés, déterminés par la construction de la vue partielle et le traitement de l'information.

a) **P2P structuré**

Dans le cas de réseaux overlay P2P structurés, les pairs sont reliés de manière étroitement contrôlée et organisée. Le nom fait principalement référence aux approches basées sur les tables DHT (Distributed Hash Tables) utilisées pour l'indexation distribuée du contenu représenté par des paires (clé, valeur). Le graphe structuré permet une découverte efficace des éléments de données (valeur) à l'aide de leurs clés (Lua, Crowcroft, Pias, Sharma, & Lim, 2005), chaque homologue du réseau recevant une partie de l'index et une table de routage dans le sous-ensemble des nœuds. Cependant, cette classe de systèmes ne supporte pas les requêtes complexes (Lua et al., 2005). De nombreux protocoles implémentant ce schéma, par exemple: Chord (Stoica et al., 2003), Tapestry (Zhao et al., 2004), Pastry (Rowstron & Druschel, 2001), Réseau adressable de contenu (CAN) (Ratnasamy, Francis, Handley, Karp, & Shenker, 2001), Kademlia (Kademlia, 2001) et Viceroy (Malkhi, Naor, & Ratajczak, 2002).

b) **P2P non structuré**

Contrairement aux overlay bien structurées créées par le groupe de modèles précédent, les réseaux overlay P2P non structurés (ou les réseaux overlay aléatoires (Van Steen, 2010)) conservent un degré élevé d'aléatoire dans la vue partielle de chaque pair. La première génération de modèles comprend les réseaux P2P purs et centralisés, la seconde - les réseaux hybrides (ou semi-centralisés) (Eberspächer & Schollmeier, 2005).

L'idée du partage de fichiers P2P a été lancée par Napster ("Napster,") en 1999 avec l'introduction du réseau P2P centralisé. Dans cette approche, une entité centrale est nécessaire pour fournir un index des pairs et des ressources partagées par eux. Cette centralisation constitue un goulot d'étranglement pour une évolutivité massive. En outre, cela crée un point d'échec unique. Ces problèmes étaient clairement visibles lorsqu'en 2001, Napster avait dû fermer son service après le procès intenté par la RIAA (Recording Industry Association of America).

Des réseaux P2P purs, tels que Gnutella (version 0.4 (Kirk, 2003)), Freenet (Clarke, Sandberg, Wiley, & Hong, 2001) ou Newscast (Jelasity & Steen, 2002), ont été développés pour résoudre les problèmes liés à la centralisation. Sans l'index central, les requêtes concernant le contenu partagé par les nœuds sont effectuées par des mécanismes d'inondation (Luke, Balan, & Panait, 2003) (la marche aléatoires, l'utilisation de TTL

(Time-to-Live), recherche d'épidémies, etc.). La topologie du graphe émergent (aléatoire) est auto-organisée, les décisions sur le routage sont prises localement sur chaque nœud. De plus, le diamètre du réseau est petit. Tout cela, ainsi que la relation de petit monde entre pairs, est la base du succès des protocoles mentionnés.

L'inondation de requêtes utilisée dans les réseaux P2P purs génère une quantité potentiellement énorme de trafic de signalisation (Eberspächer & Schollmeier, 2005). Dans le cas de Gnutella, le système s'est effondré en 2001, après le passage d'un nombre considérable d'anciens utilisateurs de Napster à ce réseau en quelques jours (Steinmetz & Wehrle, 2005). Pour contrer ce problème, des schémas hybrides (comme Gnutella 0.6 ("The Gnutella Developer Forum (GDF)," 2001)) ont été développés. En conséquence, une sorte de centralisation a été réintroduite dans le réseau, créant une hiérarchie entre les pairs. Le rôle central est partagé entre des nœuds désignés, appelés super-pairs. Ces concentrateurs ont un degré de connexion plus élevé que le reste des nœuds (feuilles). Cela ne signifie pas pour autant que l'évolutivité de la solution est réduite, car le nombre de super-pairs varie en fonction de la taille du réseau.

1.5 Les desktop grids et les systèmes de calculs volontaires (DGVCS)

Dans cette section, nous fournissons une description des projets les plus pertinents de la DGVCS, classés en fonction de leur contribution, cette catégorisation adaptée de (Barrera et al., 2012a). Nous commencerons par le premier projet de recherche documenté sur les DGVCS pour finir par les projets les plus récents et les projets en cours de développement.

1.5.1 DGVCS sur réseaux locaux LANs

Cette catégorie représente l'origine des DGCVS et se caractérise par l'utilisation des préexistantes ressources de calculs à faible distribution et hétérogénéité, connectées par un réseau local (LAN). Dans cette catégorie, nous mettons en évidence les deux anciens projets: *Worm* (Shoch & Hupp, 1982) et *Condor* (Litzkow, Livny, & Mutka, 1988).

a) Le projet Worm

Ce projet, proposé en 1978 par le Centre de recherche Xerox Palo Alto (PARC[6]), visait à développer des applications Worm capables de se reproduire dans les machines inactives.

[6] PARC: Palo Alto Research Center.

Le Worm a inclus des mécanismes pour la détection de machines inactives et pour la réplication de processus (Barrera et al., 2012a).

La contribution principale du projet Worm était de jeter les bases de l'utilisation de ressources informatiques non dédiées par la programmation d'exécutions opportunistes pendant la nuit, lorsque la plupart des ressources informatiques organisées dans le labo d'Alto pouvaient être inutilisées. La suite de tests a été réalisée à l'aide d'une infrastructure homogène comprenant 100 ordinateurs d'Alto, chacun étant connecté par un réseau local Ethernet à des serveurs de fichiers, des serveurs de boot et des serveurs DNS[7].

b) **Le projet Condor**

En 1988, l'Université du Wisconsin-Madison a lancé le projet Remote-Unix (RU) (Livny & Raman, 1998) en développant un système de gestion de charge spécialisé pour les tâches intensives appelées Condor. Comme d'autres systèmes de mise en file d'attente par lots, Condor fournit un mécanisme pour gérer une file d'attente de travail, la planification des politiques, les schémas de priorité, la surveillance et la gestion des ressources. De cette façon, les utilisateurs peuvent envoyer à Condor leur requête individuelle ou un ensemble des requêtes à planifier sur les ressources disponibles. Condor est chargé de placer les requêtes dans la file d'attente, de choisir la ressource la plus appropriée pour exécuter ces requêtes en fonction d'une stratégie de planification, d'exécuter les requêtes, de surveiller leur progression, et enfin informer l'utilisateur par les résultats achever.

1.5.2 **DGVCS à usage unique sur Internet**

Cette catégorie se caractérise par l'émergence de DGVCS basés sur l'Internet à usage individuel, avec la possibilité d'utiliser une partie des ressources informatiques existant dans les zones administratives indépendantes, hautement distribuées et hétérogènes. Ces conditions facilitaient à atteindre des capacités de traitement dans l'ordre des pétaflops, mais imposaient des restrictions sur la taille des messages et des fichiers à transférer dans le système. Cette catégorie comprend deux grands projets majeurs: GIMPS ("Mersenne, Research Inc.," 1996) et SETI@home (Anderson et al., 2002).

[7] DNS: Domain Name System.

a) **GIMPS**

Le projet GIMPS[8] est un projet de calcul distribué dédié à la recherche de nombres premiers de Mersenne, développé par Mersenne Research, Inc. GIMPS est actuellement sous licence GNU GPL (General Public License). Le projet a été fondé en janvier 1996 et est considéré comme le premier projet de calcul volontaire au monde. Son objectif principal est la recherche d'un type spécial de nombres premiers particulièrement utiles pour le développement d'algorithmes de cryptage rapide et pour les tests de performance.

b) **SETI@home**

Le projet SETI@home représente la prochaine étape importante dans la contribution aux DGVCS en permettant le calcul évolutif de millions de ressources afin de résoudre un seul problème : SETI (Search for Extraterrestial Intelligence). Le projet a été lancé au milieu de 1999 à l'Université de Californie à Berkeley, et a été considéré comme le projet bénéficiant du temps de traitement de calcul le plus long de l'histoire.

L'architecture originale de SETI@home est un modèle client/serveur évolutif sur Internet avec une dépendance directe d'un serveur centralisé très puissant, coordonnant la livraison des tâches aux clients, tout en recevant leurs résultats au travers de petits messages communiqués via HTTP (Anderson et al., 2002). Le serveur central est responsable pour la distribution des tâches hautement redondante (à un niveau double ou triple), non seulement pour gérer les déconnexions inattendues des clients, mais également pour vérifier la validité des résultats obtenus. Ce processus permet de supprimer les résultats générés sur des ordinateurs compromis (en raison d'utilisateurs malveillants, ou de problèmes matériels/logiciels).

1.5.3 **DGVCS à usage général sur Internet**

Cette catégorie se caractérise par l'émergence de DGVCS polyvalents à l'échelle Internet. Comme la catégorie précédente, ces DGVCS ont la capacité de tirer parti d'infrastructures préexistantes, hautement distribuées et hétérogènes, mais ici, les solutions ne sont pas destinées à résoudre un seul problème. Nous incluons deux projets principaux: Distributed.net ("Distributed.Net," 1997) et BOINC (Anderson, 2004).

[8] GIMPS: Great Internet Mersenne Prime Search

a) **Distributed.net**

Distributed.net est une organisation créée en 1997 sous la licence GNU FPL (Freeware Public License). Sa contribution principale est de proposer la première mise en œuvre d'un système de calcul réparti à usage général dans le monde. Le projet a été conçu pour casser les algorithmes de chiffrement et rechercher les règles optimales de Golomb (OGR[9]), particulièrement utiles pour le codage, le placement de capteurs pour la cristallographie à rayons X et pour l'étude de la radioastronomie. Les tâches sont caractérisées par l'utilisation intensive d'une énorme capacité de traitement et par leur répartition naturelle dans des unités de travail non dépendantes pour générer des résultats.

b) **BOINC**

BOINC (Berkeley Open Infrastructure for Network Computing), est une infrastructure ouverte pour le calcul en réseau qui a lancé en 2002. BOINC vise à utiliser des ressources de calculs pour le développement de projets scientifiques polyvalents (environ 30 projets scientifiques utilisent BOINC ("BOINC," 2018)), en masquant les complexités liées à la création, à l'exploitation et à la maintenance des ressources publiques des projets de calculs, en fournissant un ensemble d'outils pour la construction d'une infrastructure sécurisée avec des serveurs de domaine autonome et avec une évolutivité élevée sur Internet.

1.5.4 DGVCS dans les environnements de calcul en grille

Cette catégorie se caractérise par l'apparition de DGVCSs spécialisée dans les projets de grilles de calcul d'évolutivité variable. Semblables à la catégorie précédente, ces DGVCS ont la capacité de mobiliser des ressources publiques préexistantes. Cependant, les projets dans cette catégorie impliquent le déploiement de middlewares et un ordonnanceur pour le traitement des tâches qui nécessite une grande capacité de calcul. Cette catégorie comprend cinq projets: Bayanihan Computing.NET (Sarmenta et al., 2002), Condor-G (Frey, Tannenbaum, Livny, Foster, & Tuecke, 2002), InteGrade (Goldchleger, Kon, Goldman, Finger, & Bezerra, 2004), OurGrid (Brasileiro & Miranda, 2009) et UnaGrid (Castro, Rosales, Villamizar, & Jiménez, 2010).

[9] OGR: Optimal Golomb Rulers

a) **Bayanihan Computing.NET**

Bayanihan est apparu en 2001 en tant que cadre générique pour le calcul en grille basé sur la plateforme .NET de Microsoft. Bayanihan implémente le calcul volontaire en fournissant un service Web nommé PoolService associé à des ordinateurs qui agissent en tant que clients (volontaires) de ressources de calculs en utilisant l'architecture générique (adapté de (Sarmenta et al., 2002)).

Le service Web principal permet aux clients de créer des ensembles des tâches à calculer qui sont envoyées à des volontaires pour les exécuter et renvoient des résultats. La structure permet d'exécuter des applications au format assembleur, telles que les fichiers DLL (Dynamics Link Library), qui sont téléchargés par des volontaires (Sarmenta et al., 2002). (Les mécanismes de sécurité de la structure Microsoft. NET permettent aux ordinateurs volontaires d'exécuter ces assemblys en toute sécurité.). Contrairement à GIMPS, et SETI@home, Bayanihan permet d'exécuter des applications de terme générales. Aussi le même PoolService peut être utilisé en même temps par différents clients.

b) **Condor-G**

Comme Condor est l'un des DGVCS, ce qui lui permet de tirer parti des ressources de calculs disponibles sur des *centaines* d'ordinateurs de bureau classiques appartenant à un domaine administratif. Grâce à ces avantages, Condor a commencé depuis l'an 2000 de développer un cadre de partage et de valorisation des ressources de calculs entre différents domaines administratifs, qui a été appelé ***Condor-G***. Condor-G permet d'hériter plusieurs caractéristiques de Condor, en particulier celles liées à l'utilisation de ressources inactives dans un domaine administratif, à la disponibilité d'outils et de mécanismes de gestion et de découverte de ressources, ainsi qu'aux mesures de sécurité dans des environnements multidomaines fournis par ***Globus*** (le middleware de grille standard). Condor-G combine les protocoles de gestion de ressources multidomaines de ***Globus Toolkit*** et la gestion de ressources intradomaine de ***Condor***, ce qui permet aux utilisateurs de tirer parti des ressources de calculs inactives provenant de différents domaines administratifs, comme si tous appartenaient à un seul domaine.

c) **InteGrade**

InteGrade est une infrastructure de middleware de grille GNU LGPL (Lesser General Public License), basée sur l'utilisation opportuniste de ressources de calculs inactives. Le projet a été lancé au second semestre 2002 à l'initiative de l'Institut de mathématiques et de statistique de l'Université de Sao Paulo (EMI-USP), du département d'informatique de

l'Université pontificale catholique de Rio de Janeiro (PUCRio), l'université Federal de Goiás (UFG), le département d'informatique de l'Université fédérale de Maranhão (UFMA) et la faculté d'informatique de l'Université fédérale de Mato Grosso do Sul (UFMS) au Brésil.

La principale contribution ***d'InteGrade*** est la mise en œuvre d'un composant d'analyse du modèle d'utilisation des ressources de calculs, capable de collecter des données statistiques et de déterminer de manière probabiliste la disponibilité d'une machine, ainsi que la commodité d'attribuer un travail spécifique à une tel machine. Cette exécution évolue dans le temps, grâce à la collecte permanente de données. Basé sur ce composant, InteGrade prend en charge les applications séquentielles, paramétriques (Parameter Sweep Applications) et parallèles (Bulk Synchronous Parallel) avec des exigences de communication et de synchronisation entre les nœuds. Pour atteindre cet objectif, InteGrade intègre un composant qui analyse les caractéristiques des systèmes clients, ainsi que les connexions réseau entre eux, permettant d'établir des paramètres spécifiques pour la mise en œuvre des travaux, tels que le nombre de machines, la capacité CPU et RAM requise, la vitesse de connexion entre les nœuds, etc. Compte tenu des caractéristiques de disponibilité dynamique des infrastructures opportunistes, la prise en charge de l'application parallèle BSP suppose une stratégie du meilleur effort.

d) **OurGrid**

L'implémentation suivante de cette catégorie est OurGrid, un système de partage de ressources Open Source basé sur un réseau P2P qui facilite le partage équitable des ressources pour former des infrastructures de calcul en grille. Dans OurGrid, chaque pair représente un site entier dans un domaine administratif différent. Bien qu'il soit opérationnel depuis 2010 (Brasileiro & Miranda, 2009), le projet a débuté en 2004 et a été développés au département des systèmes et de l'informatique de l'Université fédérale de Campina Grande au Brésil. OurGrid est actuellement mis en œuvre dans le cadre du projet EELA-2 (E-science grid facility for Europe and Latin America) et vise deux objectifs principaux: le premier est de promouvoir l'agrégation évolutive des ressources de calculs vers une infrastructure de grille opportuniste nécessitant des interactions minimales avec le propriétaire de la ressource. Le deuxième objective c'est la conception d'une plate-forme ouverte, extensible et facile à installer, capable d'exécuter des applications à tâches multiples (BoT[10]) (Cirne et al., 2003). OurGrid encourage une culture de partage des ressources afin de garantir un accès juste au réseau. Le processus de partage des ressources

[10] BoT: bag-of-tasks.

est basé sur le principe de donner des cycles de calculs afin d'avoir accès à une plus grande puissance de calcul fournie en coopération avec d'autres participants du réseau.

Les participants d'OurGrid doivent considérer deux hypothèses fondamentales. La première est qu'au minimum deux pairs doivent assurer la contribution effective des ressources de calculs (on peut donc toujours obtenir des ressources d'un fournisseur différent au sein de la communauté). La deuxième est de faire face au manque de garanties de QOS[11] offertes pour les applications déployées sur OurGrid. Cette dernière caractéristique réduit les complexités associées aux modèles d'économie de réseau traditionnels (Abramson, Buyya, & Giddy, 2002; Wolski, Plank, Brevik, & Bryan, 2001), empêche les négociations entre les fournisseurs de ressources et favorise une culture de partage équitable des ressources, en suivant la stratégie du meilleur effort.

e) **UnaGrid**

UnaGrid est une virtuelle infrastructure de grille opportuniste, dont le but principal est l'utilisation efficace de ressources de calculs inutilisées par une stratégie opportuniste facilitant la tâche des applications de grille nécessitant une grande capacité de traitement. Le projet a débuté à l'initiative du Groupe des technologies de l'information et de la communication (COMIT[12]) de l'Université de los Andes en Colombie au second semestre de 2008.

L'une des contributions majeures d'UnaGrid est le concept de cluster virtuel personnalisé (Customized Virtual Cluster - CVC), qui fait référence à une infrastructure de calculs composée d'un ensemble d'ordinateurs de bureau interconnectés et conventionnels exécutant des machines virtuelles pour construire une grille de calculs, adapté au déploiement d'applications en grille. Le CVC repose sur deux stratégies de base. La première stratégie consiste à utiliser les technologies de virtualisation pour faciliter l'encapsulation d'environnements d'exécution personnalisés. Pour ce faire, les machines virtuelles doivent disposer des configurations spécifiques des systèmes d'exploitation, des middlewares, des frameworks. La deuxième stratégie repose sur l'exécution de machines virtuelles en tant que processus d'arrière-plan de priorité basse. UnaGrid accorde une attention particulière à être aussi peu intrusif que possible. L'infrastructure virtuelle déployée peut ne pas baisser la qualité de service perçue par l'utilisateur final de la

[11] QOS: quality of service.

[12] COMIT: Communications and Information Technology Group

ressource partagée, car les machines physiques utilisées sont celles des ordinateurs traditionnels des laboratoires. Les utilisateurs finaux ne sont donc pas au courant des opérations de cluster virtuel sous-jacentes.

1.5.5 Taxonomie des DGVCS

Après la description des différents DGVCS, nous présentons une taxonomie adapté de (Barrera et al., 2012a) organisée selon les caractéristiques principales suivantes: l'architecture, le niveau d'évolutivité, le type de fournisseur de ressources, l'objectif, le modèle d'application pris en charge, la plate-forme implémentant le DGVCS, la portabilité, le type de licence, et la possibilité de spécifier les ressources souhaitées (voir

Tableau 1.1).

Architecture : Les différentes composantes d'une DGVCS sont régulièrement organisées selon une approche *centralisée* ou *distribuée*. Une organisation centralisée utilise le modèle client/serveur, qui comprend des clients (utilisateurs), des fournisseurs de ressources (ressources partagées) et des serveurs (coordinateurs). Dans cette organisation, le client envoie des requêtes (travaux de calculs) à un serveur, qui les reçoit et les divise souvent en plus petites tâches. Sur la base des informations recueillies auprès de différents services de surveillance (ordonnanceurs), le serveur attribue une tâche aux fournisseurs de ressources, qui exécutent les tâches en utilisant les capacités disponibles. Une fois la tâche terminée, le fournisseur de ressources renvoie les résultats au serveur, qui les vérifie éventuellement avant de les envoyer au client. Les organisations distribuées peuvent être classées en deux sous-catégories, celles utilisant un système d'égal à égal (*P2P*) et celles utilisant une approche *hiérarchique*. Dans une organisation P2P, il existe des clients et des fournisseurs de ressources, mais il n'y a pas de serveur centralisé. Les fournisseurs de ressources ont une vue partielle du système et gèrent une stratégie de planification distribuée. Les clients envoient des requêtes de travail directement à un fournisseur « proche » (le fournisseur principal ou maître de ce client), qui utilise la stratégie de planification distribuée pour attribuer des travaux à d'autres fournisseurs de ressources. Les fournisseurs exécutant les travaux renvoient les résultats à ce fournisseur maître qui les vérifie et les renvoie au client demandeur. Dans les DGVCS basés sur le modèle hiérarchique, les fournisseurs de ressources sont organisés de manière à ce qu'un DGVCS puisse envoyer une demande de travail à d'autres DGVCS avec les ressources disponibles. Pour ce faire, vous créez une hiérarchie DGVCS dans laquelle les DGVCS de haut niveau envoient des travaux à des DGVCS de niveau inférieur, lorsque les capacités de calcul de

ces derniers ne sont pas pleinement utilisées. Pour ce faire, les DGVCS sont configurés pour participer à une hiérarchie, connectant chaque DGVCS à un DGVCS de niveau supérieur, permettant ainsi aux DGVCS de haut niveau de considérer les DGVCS inférieurs comme des principaux fournisseurs de ressources de calculs.

Évolutivité : En fonction de leur évolutivité, les DGVCS peuvent être classés en tant que DGVCS pour les réseaux locaux (LAN) ou DGVCS pour Internet. Les DGVCS pour réseaux locaux cherchent à exploiter les ressources de calcul d'une organisation ou d'une institution ; dans ce cas, les ressources de calculs appartiennent régulièrement à un seul domaine administratif, ce qui confère à ces systèmes une connectivité plus stable et fiable, réduit les risques liés aux problèmes de sécurité et un degré de contrôle élevé sur les ressources de calculs offerts. D'autre part, les DGVCS pour Internet recherchent des ressources de calculs anonymes réparties géographiquement et traitent des problèmes de communications à faible bande passante (pare-feu, NAT[13], etc.), des ressources malveillantes et des problèmes liés aux intrusions, ce qui implique des risques de sécurité élevés, la communication non fiable, et une disponibilité réduite des ressources. Bien que les DGVCS pour les réseaux locaux tirent certains avantages d'un contrôle accru des ressources partagées ainsi que d'une amélioration de la disponibilité et de la sécurité, ils sont limités à l'utilisation des ressources disponibles au sein d'une organisation ou d'une institution. C'est pourquoi les DGVCS pour Internet permettent de regrouper des capacités de calcul au niveau de milliers voire de millions d'ordinateurs connectés via Internet. Mais le prix à payer : la sécurité, la fiabilité et la disponibilité.

Type de fournisseur de ressources : En tenant en compte la manière dont les ordinateurs fassent partie d'une DGVCS et fournissent leurs ressources de calculs, les DGVCS peuvent être classés en fonction du type de fournisseurs de ressources volontaires ou institutionnels. Les DGVCS avec fournisseurs volontaires obtiennent leurs capacités de calculs à partir des ordinateurs dont les propriétaires/utilisateurs finaux décident volontairement de donner leurs ressources inutilisées à un DGVCS (ces ordinateurs doivent avoir un accès internet). Les DGVCS avec fournisseurs institutionnels sont ceux qui obtiennent des ressources de calculs à partir d'ordinateurs, qui sont intégrées au système par un administrateur système institutionnel. Souvent associés aux réseaux locaux, ces DGVCS tirent parti des ressources de calculs sous-utilisées pendant que le personnel de l'entreprise effectue les activités quotidiennes.

[13] NAT: Network Address Translation

Les utilisateurs finaux des fournisseurs de ressources étant normalement identifiés.

Objectif : Les DGVCS peuvent être classés en tant que DGVCS à usage *unique* ou à usage *général*. Les DGVC à usage unique exploitent leurs capacités de calculs pour résoudre un seul problème spécifique. Ils sont régulièrement administrés par une seule organisation. Les DGVCS à usage général permettant de résoudre différents types de problèmes ; ils sont régulièrement administrés par plusieurs organisations, appelées organisations virtuelles, cherchant à résoudre différents problèmes par le déploiement de multiples applications.

Type d'application : selon les applications à exécuter, les DGVCS peuvent être regroupés en deux catégories principales : le modèle maître/travailleur qui exécute les tâches indépendantes et le modèle de programmation parallèle qui nécessite une communication entre les processus. Dans le modèle basé sur maître/travailleur, un nœud maître (serveur) envoie un ensemble de tâches indépendantes à un ensemble de nœuds travailleurs. Le nœud maître attend que chacun des nœuds travailleurs exécute sa portion de travail et renvoie son résultat. Les tâches exécutées sur chaque nœud travailleur sont totalement indépendantes et peuvent être exécutées en parallèle sur différents travailleurs. Le type d'application en général est des applications parallèles dont les tâches sont indépendantes (Bag of Task applications). Dans le modèle de programmation parallèle, plusieurs processus coopèrent entre eux pour exécuter une tâche commune. Cette coopération est réalisée au moyen de communications utilisant différents paradigmes de programmation parallèle tels que MPI (Message Passing Interface), PVM (Parallel Virtual Machine) ou BSP (Bulk Synchronous Parallel). Dans ces schémas, une tâche est effectuée via l'exécution de plusieurs processus s'exécutant sur des ordinateurs différents. Dans de telles applications, la priorité devrait être donnée à différents problèmes tels que la synchronisation entre processus, la transmission de messages, l'accès à distance à la mémoire, les retards de communication, entre autres.

Plate-forme : En fonction de la plate-forme utilisée par les fournisseurs de ressources de calculs, les DGVCS peuvent être classés en trois catégories : middleware, Web et virtualisation.

Les DGVCS basés sur les middlewares se caractérisent par la nécessité d'installer un middleware spécifique sur le système d'exploitation des ressources. Ce middleware permet aux applications DGVCS d'être exécutées sur le système des ressources.

Dans les DGVCS basés sur le Web, les applications doivent être développées en Java et mises à disposition dans une page Web. Les fournisseurs de ressources accèdent à cette page Web via un navigateur standard et exécutent le code en tant qu'applet.

Certains DGVCS utilisent des machines virtuelles pour faciliter et accélérer l'installation, la configuration et le déploiement des applications nécessaires pour tirer parti des ressources de traitement inactives et pour augmenter leur portabilité. Dans ces DGVCS, les fournisseurs de ressources ont régulièrement installé un outil de virtualisation tel que VMware ("VMware Inc.," 2008), Virtual Box ("Sun Microsystems Inc.," 2007), etc. Les ressources sont configurées de sorte qu'elles implémentent des stratégies de partage spécifiques et qu'une image d'une machine virtuelle contenant tous les logiciels (logiciels, bibliothèques, middleware et applications) requis pour s'exécuter dans le contexte DGVCS est stockée localement. Une fois configurée, la machine virtuelle s'exécute en fonction de paramètres configurés et commence à faire partie du système pour exécuter les différentes tâches.

Portabilité : la portabilité sur DGVCS est liée à la capacité de déploiement sur les différents systèmes d'exploitation des ressources accordés. Dans les DGVCS dépendant du système d'exploitation, la quantité de ressources disponibles est limitée aux ressources qui utilisent le système d'exploitation sur lequel le logiciel DGVCS fonctionne. Au contraire, les DGVCS indépendants du système d'exploitation peuvent regrouper davantage de ressources de calculs en utilisant l'une des stratégies suivantes: 1) utiliser un langage indépendant du système d'exploitation, tel que Java; 2) créer et compiler le code source pour chacun des systèmes d'exploitation sur lesquels le DGVCS est censé fonctionner; et 3) utiliser des outils de virtualisation pour permettre à DGVCS de s'exécuter sur un logiciel de virtualisation particulier plutôt que sur un système d'exploitation spécifique..

Licence : les DGVCS peuvent être classés comme propriétaires ou Open Source.

Spécification des ressources : l'une des stratégies utilisées dans les DGVCS est la possibilité de faire correspondre les tâches et les ressources. Il ouvre deux catégories, celles qui permettent aux utilisateurs de spécifier les ressources requises pour l'exécution de leurs tâches et celles qui offrent le même type de ressources, quel que soit le type de travail à exécuter. Dans la première catégorie, la spécification est souvent exprimée par une commande. Lors de l'envoi d'un travail, l'utilisateur spécifie les ressources requises et le planificateur DGVCS utilise ces informations pour sélectionner les ressources appropriées.

Tableau 1.1 Taxonomie DGVCS.

DGVCSs	Architecture	L'évolutivité sur	Fournisseur de ressources	Objectif	Type d'application	Plate-forme	Portabilité Méthode	Licence	Inclure la spécification des ressources
Worm (Shoch & Hupp, 1982)	Centralisé	LAN	Institution	Général	BoT	Middleware	Sans système d'exploitation	Propriétaire	✘
Condor (Litzkow et al., 1988)	Centralisé	LAN	Institution	Général	BoT/ Parallèle	Middleware	Compilé pour chaque système d'exploitation	Open source	✓
GIMPS ("Mersenne, Research Inc.," 1996)	Centralisé	Internet	Volontaire	Unique	BoT	Middleware	Compilé pour chaque système d'exploitation	Open source	✘
SETI@Home (Anderson et al., 2002)	Centralisé	Internet	Volontaire	Unique	BoT	Middleware	Compilé pour chaque système d'exploitation	Open source	✘
Distributed.Net ("Distributed.Net," 1997)	Hiérarchique	Internet	Volontaire	Général	BoT	Middleware	Compilé pour chaque système d'exploitation	Open source	✘

BOINC (Anderson, 2004)	Hiérarchique	Internet	Volontaire	Général	BoT	Middleware	Compilé pour chaque système d'exploitation	Open source	✓
Bayanihan Computing.NET (Sarmenta et al., 2002)	Centralisé	LAN	Volontaire	Général	BoT	Web	Tout système d'exploitation	Open source	✘
Condor-G (Frey et al., 2002)	Hiérarchique	Internet	Institution	Général	BoT	Middleware	Compilé pour chaque système d'exploitation	Open source	✓
InteGrade (Goldchleger et al., 2004)	Hiérarchique	Internet	Institution	Général	BoT/ Parallèle	Middleware	Java environnement	Open source	✓
OurGrid (Brasileiro & Miranda, 2009)	P2P/ Centralisé	Internet	Volontaire	Général	BoT	Middleware /Virtualisation	Virtualisation	Open source	✓
UnaGrid (Castro et al., 2010)	Hiérarchique	LAN	Institution	Général	BoT	Virtualisation	Virtualisation	Open source	✘

1.6 Conclusion

Il existe quatre grandes catégories de systèmes pour les calculs distribués : les clusters, les grilles (basées sur les clusters), les desktop grids et les systèmes de calcul volontaire (DGVCS), et le cloud. Chacune d'entre elles avec des différentes propriétés de ressources disponibles. Les réseaux overlay déterminent comment les tâches constituant un calcul distribué interagissent les unes avec les autres.

La dernière section présente un état de l'art sur les DGVCS, où nous mettons en exergue les projets les plus pertinents visant à fournir des stratégies permettant de tirer parti des cycles de calculs inactifs. Les DGVCS ont permis le partage de ressources de calculs distribuées même si ces ressources sont utilisées pour d'autres tâches. Cette stratégie est devenue très utile, car elle a permis d'agréger des millions de ressources de calculs distribuées par des volontaires, ce qui représente une solution efficace pour soutenir différents projets de science en ligne, dans différents domaines. Dans le chapitre suivant nous présenterons les concepts de base sur la tolérance aux pannes, ainsi que l'équilibrage de charge. Ensuite, nous citons quelques travaux récents réalisés pour résoudre : le problème d'équilibrage de charge et la tolérance aux pannes dans les déférents systèmes distribués.

Chapitre 2 LA TOLERANCE AUX PANNES ET L'EQUILIBRAGE DE CHARGE

2.1 Introduction

Les systèmes distribués sont des groupes de nœuds interconnectés, qui ont le même objectif. Afin de répondre à un problème spécifique, la tâche a été divisée en plusieurs petites tâches allouées à ces nœuds. Chaque nœud exécute sa partie de travail et soumet à nouveau les résultats au nœud de soumission. De plus, les systèmes distribués peuvent être homogènes (cluster), hétérogènes (Grid, Cloud et DGVCS). Ils sont soumis à différents types de problèmes tels que ***la qualité de service*** (QoS), ***la sélection des ressources***, ***l'équilibrage de la charge*** et ***la tolérance aux pannes***. La tolérance aux pannes perçoit la réaction d'un système à une défaillance surprenante de matériel/logiciel, par rapport à un système centralisé ; il est difficile de détecter les échecs dans le système distribué. La tolérance aux pannes est principalement composée de deux éléments de base, *la détection* et *la récupération d'échec*.

Dans ce qui suit, nous présenterons dans la première partie les concepts de base sur la tolérance aux pannes, type des défauts, les modèles de défaillance, et les différentes techniques de tolérance aux pannes dans le paradigme distribué telles que les nouvelles tentatives (Retry), la réplication, *l'équilibrage de charge*, les points de contrôle (Checkpointing) et la journalisation des messages (Message logging), etc. Dans la seconde partie, nous citerons en détail la technique de l'équilibrage de charge avec ses avantages, et nous discutons en détail les différents algorithmes trouvés pour les deux approches dynamique et statique dans la littérature. À la fin de ce chapitre, nous citons quelques travaux récents réalisés pour résoudre : le problème d'équilibrage de charge et la tolérance aux pannes dans les déférents systèmes distribués.

2.2 Concepts de base sur la tolérance aux pannes

Afin de comprendre le rôle principal de la tolérance aux pannes dans les systèmes distribués, nous devons d'abord vérifier ce que cela signifie réellement de tolérer les pannes ou d'être toléré par les pannes pour un système distribué. Cette définition est très proche de « Systèmes fiables ». Comme le montre la Figure 2.1, la fiabilité est un mot-clé qui inclut de nombreuses exigences utiles pour les systèmes distribués. Dans (Andrew S. Tanenbaum & Steen, 2017), les auteurs présentent les principales caractéristiques de la sûreté de fonctionnement en quatre attributs (la disponibilité, la fiabilité, la sécurité, la

maintenabilité), et pour assurer un degré élevé de sécurité , en parle d'un problème d'intégrité. Les auteurs en (Avizienis, Laprie, Randell, & Landwehr, 2004) ont ajouté le cinquième attribut (l'intégrité):

2.2.1 La disponibilité

La disponibilité est définie comme la propriété qu'un système est prêt à être utilisé immédiatement. En général, il fait référence à la probabilité que le système fonctionne correctement à un moment donné et qu'il soit disponible pour exécuter ses fonctions pour le compte de ses utilisateurs. En d'autres termes, un système hautement disponible est celui qui fonctionnera le plus probablement à un instant donné (Andrew S. Tanenbaum & Steen, 2017).

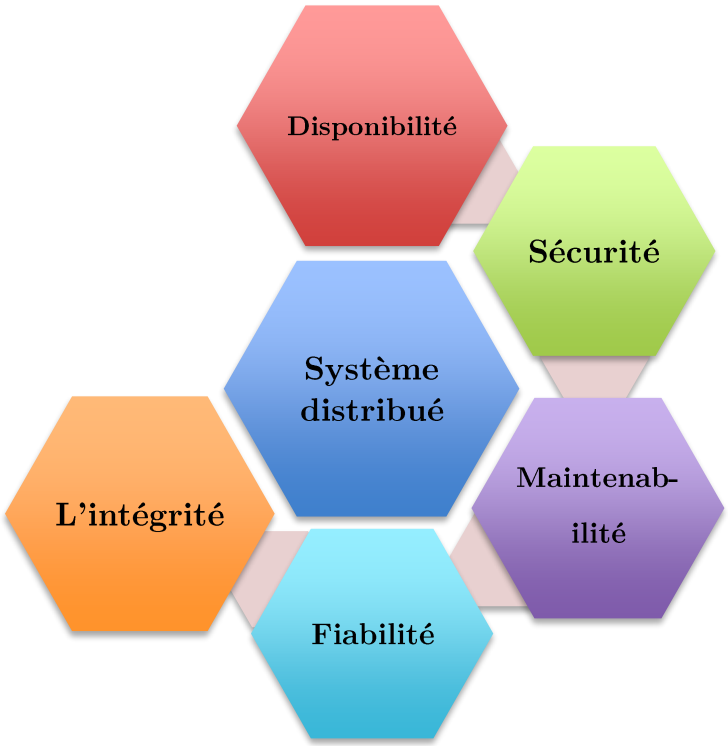

Figure 2.1 Un Systèmes distribués fiables.

2.2.2 La fiabilité

La fiabilité fait référence à la propriété qu'un système peut fonctionner en continu sans défaillance. Contrairement à la disponibilité, la fiabilité est définie en termes d'intervalle de temps au lieu d'un instant. Un système très fiable est un système qui continuera très probablement à fonctionner sans interruption pendant une période relativement longue. Si un système tombe en panne en moyenne pendant une heure, apparemment une

milliseconde au hasard, toutes les heures, sa disponibilité dépasse les 99,9999%, mais elle n'est toujours pas fiable (Andrew S. Tanenbaum & Steen, 2017). De même, un système qui ne tombe jamais en panne, mais qui est arrêté pendant deux semaines spécifiques en un mois au hasard, offre une fiabilité élevée, mais une disponibilité de seulement 96 %. Les deux ne sont pas les mêmes. (Bref la fiabilité est que le système peut fonctionner en continu sans défaillance).

2.2.3 La sécurité

Lorsqu'un système échoue temporairement à fonctionner correctement, aucun événement catastrophique ne se produit. Par exemple, de nombreux systèmes de contrôle de processus, tels que ceux utilisés pour contrôler les centres nucléaires ou d'envoyer des personnes à l'espace, sont obligés d'avoir un haut niveau de sécurité (Andrew S. Tanenbaum & Steen, 2017). De nombreux exemples du passé (et probablement de nombreux autres à venir) montrent combien il est difficile de construire des systèmes sûrs. (Bref si un système tombe en panne, rien de catastrophique ne se produira).

2.2.4 La maintenabilité

La maintenabilité fait référence à la facilité avec laquelle un système défaillant peut être réparé. Un système hautement maintenable peut également afficher une grande disponibilité, en particulier si les défaillances peuvent être détectées et réparées automatiquement. (lorsqu'un système tombe en panne, il peut être réparé facilement et rapidement (et, parfois, sans que ses utilisateurs ne remarquent la panne)).

2.2.5 L'intégrité

L'intégrité fait référence qu'aucune altération accidentelle ou malveillante de l'information (data) n'a été effectuée même par des entités autorisées (l'absence de modifications incorrectes du système) (Andrew S Tanenbaum & Van Steen, 2007).

2.3 Qu'est-ce qu'un « échec » ?

On dit qu'un système *échoue* lorsqu'il ne peut pas tenir ses promesses. En particulier, si un système distribué est conçu pour fournir à ses utilisateurs un certain nombre de services, le système a échoué lorsqu'un ou plusieurs de ces services ne peuvent pas être (complètement) fournis.

Une *erreur* est une partie de l'état d'un système susceptible de provoquer une panne. Par exemple, lors de la transmission de paquets sur un réseau, il faut s'attendre à ce que

certains paquets aient été endommagés lorsqu'ils arrivent au destinataire. Endommagé dans ce contexte signifie que le récepteur peut lire de manière incorrecte une valeur de bit (par exemple, lire un 1 au lieu d'un 0) ou même être incapable de détecter que quelque chose est arrivé.

La cause d'une erreur s'appelle une *faute*. De toute évidence, il est important de déterminer la cause d'une erreur. Par exemple, un mauvais support de transmission peut facilement endommager des paquets. Dans ce cas, il est relativement facile de supprimer le défaut.

Une erreur peut survenir en cas de déviation du système par rapport à l'opération requise. Cette transition est appelée une activation de faute, c'est-à-dire qu'une faute en sommeil (ne produisant aucune erreur) devient active. Une erreur est détectée si sa présence est indiquée par un message ou un signal d'erreur, alors que les erreurs présentes, mais non détectées sont appelées des erreurs *latentes*. Des erreurs dans le système peuvent provoquer un échec (de service) et, en fonction de son type, des erreurs et des défauts successifs peuvent être introduits (propagation erreur/échec).

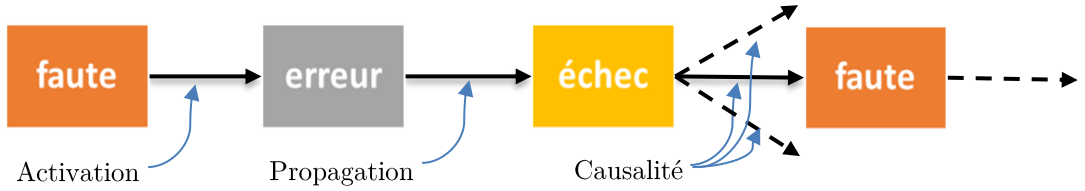

Figure 2.2 Relation entre les fautes, les erreurs et les échecs (Avizienis, Laprie, Randell, Landwehr, & computing, 2004)

La distinction entre les fautes, les erreurs et les défaillances (voir la Figure 2.2) est importante, car ces termes créent des limites permettant l'analyse et la gestion de menaces différentes. En substance, les défauts (échecs) sont la cause d'erreurs (reflétées dans l'état) qui sans traitement approprié, peuvent conduire à des échecs (résultat erroné et inattendu). Suivant ces définitions, «la tolérance au panne est la capacité d'un système à se comporter de manière bien définie une fois qu'une erreur se produit» (Avizienis et al., 2004).

2.4 Type des défauts

Les défauts sont généralement classés en transitoires, intermittents ou permanents (voir la Figure 2.3) :

2.4.1 Un défaut transitoire

Ces défauts se produisent une fois puis disparaissent. Si l'opération est répétée, le défaut disparaît. Un oiseau volant dans le faisceau d'un émetteur à micro-ondes peut causer la perte de bits sur certains réseaux (sans parler d'un oiseau grillé).

2.4.2 Un défaut intermittent

Ce défaut se produit puis disparaît automatiquement à plusieurs reprises. Les défauts intermittents causent beaucoup d'aggravations, car ils sont difficiles à diagnostiquer. Généralement, une fois traité, le système fonctionne correctement.

2.4.3 Un défaut permanent

C'est un défaut qui continue d'exister jusqu'à ce que le composant défectueux soit remplacé. Les puces épuisées, les bugs logiciels et les crashs de têtes de disque sont des exemples de pannes permanentes.

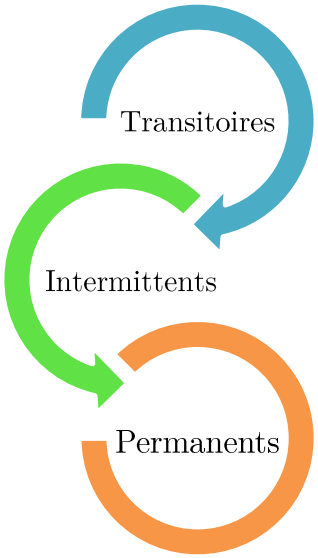

Figure 2.3 Type des défauts.

2.5 Modèles de défaillance

Les défaillances peuvent également être caractérisées en fonction de leur *domaine*, de leur *détectabilité*, de leur *cohérence,* et de leurs *conséquences* (Avizienis et al., 2004) (voir Figure 2.4 pour une description bien détaillée). Bien que ces classes et leurs sous-types soient relativement généraux, il existe des modèles de pannes spécifiques pertinents en

calcul distribué, à savoir: *crash*, *omission*, *duplication*, *synchronisation* et *défaillances byzantines* (Andrew S. Tanenbaum & Steen, 2017).

- La défaillance de crash se produit en quatre variantes, chacune associée à sa persistance. Les défaillances transitoires de crash correspondent au redémarrage du service:
  1. ***Amnesia-crash***: le système est restauré dans un état initial prédéfini, indépendant des entrées précédentes).
  2. ***Partial-amnesia-crash:*** une partie du système reste dans l'état précédant l'incident. Le reste du système est réinitialisé aux conditions initiales).
  3. ***Pause-crash:*** le système est restauré dans l'état où il se trouvait avant le crash.
  4. ***Halt-crash :*** c'est un échec permanent rencontré lorsque le système ou le service n'est pas redémarré et reste instable.
- ***Les échecs de duplication:*** se produisent dans la situation opposée: un message est envoyé ou reçu plusieurs fois.
- ***Les échecs de synchronisation:*** surviennent lorsque les contraintes de temps concernant l'exécution du service ou la livraison des données ne sont pas respectées. Ce type ne se limite pas aux retards, car une prestation de service trop précoce peut également être indésirable.
- Le dernier modèle ***l'échec byzantin*** (également appelé arbitraire) - couvre toutes les réponses (très souvent inattendues et incohérentes) d'un service ou d'un système à des moments arbitraires. Dans ce cas, les défaillances peuvent survenir périodiquement avec des résultats variables, une portée, des effets, etc. Il s'agit du type de défaillance le plus générale et le plus grave (Andrew S. Tanenbaum & Steen, 2017).
- Les échecs *d'omission* et de *duplication* sont liés à des problèmes de communication.

  Send-omissions: correspond à une situation dans laquelle un message n'est pas envoyé.

  Receive-omission: quand un message n'est pas reçu.

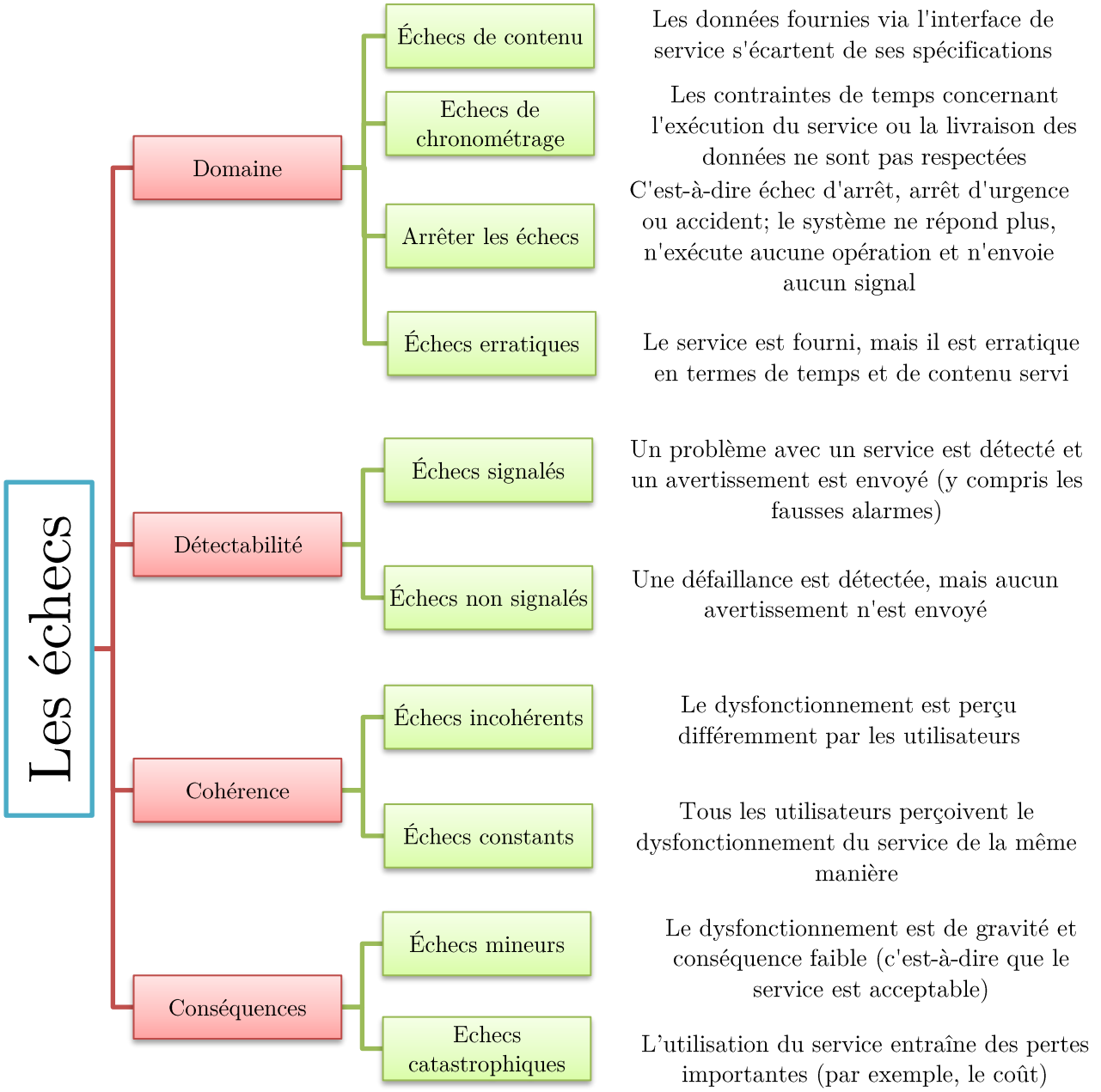

Figure 2.4 Les échecs de service (Avizienis, Laprie, Randell, & Landwehr, 2004).

2.6 Techniques de tolérance aux pannes

Les techniques de tolérance aux pannes peuvent être divisées en deux catégories principales complémentaires (Avizienis et al., 2004):

1. La détection d'erreur.
2. La récupération.

2.6.1 La détection d'erreur

La détection d'erreur peut être effectuée pendant le fonctionnement normal du service ou pendant sa suspension. La première approche de cette catégorie - **la détection simultanée** - est basée sur divers tests effectués par des composants (logiciels et/ou matériels) impliqués dans une activité particulière ou par des éléments spécialement conçus pour cette fonction. Par exemple, un composant (logiciel) peut calculer et vérifier des totaux de contrôle pour les données qu'il traite. D'autre part, un pare-feu est une bonne illustration d'un matériel (ou logiciel) désigné vers la détection des intrusions et autres activités malveillantes. **La détection préemptive** est associée à la maintenance et au diagnostic d'un système ou d'un service. Dans cette approche, l'accent est mis sur l'identification des fautes latentes et des erreurs dormantes. Cette opération peut être effectuée au démarrage du système, au démarrage d'un service ou lors de sessions de maintenance spéciales.

Après la détection d'une erreur, les méthodes de récupération sont appliquées. Selon le type de problème, des techniques de gestion des fautes ou des erreurs sont utilisées. Le premier groupe est axé sur l'élimination des erreurs de l'état du système, tandis que le second est conçu pour empêcher l'activation des erreurs (Avizienis et al., 2004). Généralement, le traitement des erreurs est résolu par :

2.6.2 La restauration

Le système est restauré dans le dernier état connu sans erreur. L'approche dépend ici d'une méthode utilisée pour suivre les changements d'état (Elnozahy, Alvisi, Wang, & Johnson, 2002). Une technique bien connue est **le point de contrôle** *(checkpoint)* - l'état d'un système est sauvegardé périodiquement (l'instantané d'un processus est stocké sur un disque fiable et stable) en tant que point de récupération potentiel dans l'avenir. De toute évidence, cette solution n'est pas simple dans le cas des systèmes distribués ou de nombreux facteurs doivent être pris en compte. Dans un tel environnement, les points de contrôle peuvent être coordonnés ou non - avec des différences de fiabilité et du coût de

synchronisation des composants distribués (pour plus de détails, voir: (Coulouris, 2011; Hwang et al., 2013)).

La restauration peut également être implémentée via **la journalisation des messages** *(message logging)*. Dans ce cas, la communication entre les composants est suivie plutôt que leur état. En cas d'erreur, le système est restauré en rejouant les messages historiques, ce qui lui permet d'atteindre un état de cohérence globale (Andrew S. Tanenbaum & Steen, 2017). Parfois, les deux techniques sont traitées comme une seule technique.

a) **Rouler vers l'avant**

L'état actuel du système erroné est supprimé. Il est néanmoins remplacé par un nouvel état créé et initialisé.

2.6.3 La compensation

Les solutions basées sur la *redondance* et la *réplication* de composants, parfois appelées *masquage de pannes*. Dans le premier cas, des composants supplémentaires (généralement du matériel) sont gardés en réserve (Hwang et al., 2013). Si des défaillances ou des erreurs se produisent, elles sont utilisées pour compenser les pertes. Par exemple, une connexion à Internet d'une plate-forme en nuage doit être basée sur les solutions d'au moins deux fournisseurs de services Internet différents (ISPs[14]).

2.6.4 La réplication

La réplication est basée sur la dispersion de plusieurs copies des composants de service. Un schéma avec répliques utilisé uniquement à des fins de tolérance aux pannes est appelé *réplication passive* (sauvegarde primaire) (Hwang et al., 2013). D'autre part, la *réplication active* se produit lorsque les répliques participent à la fourniture du service, ce qui permet d'améliorer les performances et l'applicabilité des techniques d'équilibrage de charge. La cohérence est le principal défi ici, et diverses approches sont utilisées pour le soutenir. Par exemple, les protocoles de lecture-écriture sont cruciaux dans la réplication active, car toutes les répliques doivent avoir le même état. Un autre exemple intéressant à noter est clairement visible sur les plateformes de calculs volontaires. Une stratégie de sélection appropriée de la réponse de service correcte est nécessaire lorsque les répliques renvoient

[14] ISPs: Internet Service Provider

des réponses différentes, c'est-à-dire qu'une méthode permettant d'atteindre le consensus du quorum est requise (Hwang et al., 2013).

Ces techniques ne sont pas exclusives et peuvent être utilisées ensemble. Si le système ne peut être restauré à un état correct grâce à la compensation, une restauration peut être tentée. Si cela échoue, nous pouvons utiliser la fonction de Rollforward.

Les méthodes ci-dessus peuvent être appelées techniques à usage général. Ces solutions sont relativement génériques, ce qui facilite leur mise en œuvre pour presque tous les calculs distribués. Il est également possible de déléguer la responsabilité de la tolérance aux pannes au service (ou à l'application) lui-même, ce qui permet d'adapter la solution à des besoins spécifiques, créant ainsi une approche spécifique à l'application.

Les techniques de traitement des erreurs sont appliquées après que le système a été restauré dans un état exempt d'erreur (à l'aide des méthodes décrites ci-dessus). L'objectif étant maintenant d'empêcher l'activation future de défauts détectés, quatre sous-groupes peuvent être créés (Avizienis et al., 2004) en fonction de l'intention de l'opération :

1. **Diagnostic :** les erreurs sont identifiées et leur source est localisée.
2. **Isolation :** Les composants défectueux sont séparés logiquement ou physiquement et exclus du service.
3. **Reconfiguration :** si des composants redondants sont disponibles, les composants précédemment supprimés sont remplacés par ceux-ci ; sinon, le service/la plate-forme est reconfiguré pour contourner les éléments défectueux.
4. **Réinitialisation :** la configuration du système (y compris son état) est adaptée aux nouvelles conditions (composants supprimés, ajoutés ou mis à jour).

La construction d'un système fiable est fortement liée au contrôle des erreurs. Comme expliqué par (Avizienis et al., 2004), une distinction peut être faite entre prévenir, tolérer, éliminer et prévoir les fautes. Pour nous, le problème le plus important est *la tolérance aux pannes*, ce qui signifie qu'un système peut fournir ses services même en présence de pannes.

2.7 L'équilibrage de charge

L'équilibrage de charge est l'un des problèmes principaux à résoudre dans les systèmes de calculs distribués. Afin de répondre aux attentes des utilisateurs en termes de performances et d'efficacité, un système de calculs distribué à besoin d'algorithmes d'équilibrage de charge efficaces pour la répartition de la charge de travail du système sur les ressources disponibles. L'équilibrage de la charge dans l'environnement des systèmes de calculs distribués à un impact important sur les performances (S. El-Zoghdy & Ghoniemy, 2014b; Jain & Gupta, 2009; Manekar, Poundekar, Gupta, & Nagle, 2012). Par exemple, dans le but de réduire la durée d'exécution des programmes ou des services, les programmes ou les processus du serveur peuvent être migrés de nœuds très chargés (surchargé) vers des nœuds peu chargés (sous-chargé). Ce type de migration s'appelle migration de chargement (load migration). Le nœud lourdement chargé s'appelle un émetteur, et le nœud légèrement chargé un récepteur. Les techniques d'équilibrage de charge pour les systèmes de calculs distribués ont toujours des schémas d'équilibrage de charge qui spécifient les règles d'équilibrage de charge (comment décider quel nœud est un émetteur ou un récepteur pour une migration de charge), et une architecture (comment les nœuds sont organisés pour cet équilibrage). Nous examinons les schémas en fonction de ces deux côtés.

Un schéma d'équilibrage de charge se compose de trois phases (Khan, Singh, Alam, & Saxena, 2011): Une collecte d'informations, puis une prise de décision, et en fin une migration de données.

a) La collecte d'informations

Au cours de cette phase, l'équilibreur de charge collecte les informations relatives à la répartition de la charge de travail et à l'état de l'environnement du système, et détecte s'il existe un déséquilibre de charge.

b) La prise de décision

Cette phase se concentre sur le calcul d'une distribution optimale des données.

c) La migration de données

Cette phase transfère la charge de travail excédentaire d'un nœud surchargé vers un nœud sous-chargé.

2.7.1 Avantages de l'équilibrage de charge

Certains des principaux avantages de l'équilibrage de charge sont mentionnés dans (S. El-Zoghdy & Ghoniemy, 2014b; Manekar et al., 2012):

- Réduit le temps d'attente des tâches.
- Minimise le temps de réponse des tâches.
- Il maximise l'utilisation des ressources du système.
- Il maximise le débit du système.
- Améliore la fiabilité et la stabilité du système.
- Il accepte les modifications futures.
- La longue famine est évitée pour les petits travaux.
- Atteindre la performance globale du système, en améliorant la performance de chaque nœud.

2.7.2 Politiques d'équilibrage de la charge

Pour choisir un bon algorithme d'équilibrage de charge, l'auteur (G. Sharma, 2013) a défini certaines règles de base qui sont les suivantes:

- ✓ Stratégie d'informations : spécifie quelles informations de charge de travail doivent être collectées, mises à jour, à quel moment et de quel endroit.
- ✓ Stratégie de déclenchement : détermine la période appropriée pour démarrer une opération d'équilibrage de charge.
- ✓ Stratégie de type de ressource : récupérer la tâche à partir d'une ressource la plus surchargée ou non.
- ✓ Stratégie de localisation : utilise les résultats de la stratégie de type de ressource pour trouver un partenaire approprié pour un serveur ou un récepteur.
- ✓ Politique de sélection : définit les tâches à migrer des ressources les plus occupées (source) vers les ressources les plus inactives (récepteur). Une politique de sélection prend en compte plusieurs facteurs lors de la sélection d'une tâche, par exemple : transfert de la petite tâche prendra moins de frais généraux.

2.7.3 Approches d'équilibrage de la charge

En équilibrage de charge, il existe deux approches ou chacune a ses propres algorithmes. Les deux approches sont les suivantes (Ali & Khan, 2012; Hussain et al., 2013; Jain & Gupta, 2009) *Static Load Balancing (SLB)* et *Dynamic Load Balancing (DLB)* :

a) L'approche d'équilibrage de charge statique

Dans l'approche d'équilibrage de charge statique, les décisions sont prises de manière *déterministe* ou *probabiliste* avant l'exécution de l'algorithme en fonction des performances des nœuds de calcul et restent constantes pendant l'exécution. Le nombre de tâches dans chaque nœud est fixé dans cette approche (S. El-Zoghdy & Ghoniemy, 2014a). Les méthodes d'équilibrage de charge statique ne sont pas préemptives, c'est-à-dire qu'une fois que la charge est allouée au nœud, elle ne peut pas être migrée vers un autre nœud, et ne collectent aucune information sur les nœuds de calcul (Ali & Khan, 2012; Hussain et al., 2013; Jain & Gupta, 2009). L'attribution des tâches aux nœuds de calculs est effectuée en fonction de nombreux facteurs, tels que l'étendue des ressources nécessaires, les communications interprocessus. Ces facteurs doivent être mesurés avant l'exécution du l'algorithme SLB.

Comme expliqué précédemment, les décisions d'équilibrage de charge dans les algorithmes SLB ne sont pas affectées par les informations collecter d'état du système ; au lieu de cela, ils ne sont affectés que par les informations statistiques du système. En conséquence, les algorithmes SLB présentent des avantages en analyse mathématique et en implémentation en raison de leur simplicité (Dong & Akl, 2006). En l'absence de migration des tâches au moment de l'exécution, la surcharge du système est minimisée ou supprimée (Kameda, Li, Kim, & Zhang, 2012). Un inconvénient général des algorithmes SLB est que la sélection finale d'un nœud de calcul pour l'attribution de tâche est faite lors de la création de la tâche et ne peut pas être modifiée pendant l'exécution du processus (Ali & Khan, 2012; Hussain et al., 2013).

Parmi les divers algorithmes proposés, les principaux algorithmes d'équilibrage de charge statique sont les suivants :

(a) Algorithme de Round Robin (Round Robin Algorithm)

L'algorithme Round Robin (RRA[15]) attribue les tâches de manière séquentielle et uniforme à tous les nœuds. Toutes les tâches sont attribuées à des nœuds de calculs en fonction de l'ordre Round Robin, ce qui signifie que le choix des nœuds de calcul est effectué en série et revient au premier nœud de calcul si le dernier nœud de calcul a été atteint (S. El-Zoghdy & Ghoniemy, 2014a, 2014b) (lorsque le nombre des tâches est supérieur au nombre

[15] RRA: Round Robin Algorithm

de nœud de calcul). Chaque nœud maintient son index de charge localement, indépendamment des allocations du nœud distant.

Le principal avantage de l'RRA est que la communication entre processus n'est pas requise. En règle générale, les performances RRA ne sont pas satisfaisantes, car, lorsque les tâches ont un temps de traitement inégal ou les nœuds ont des capacités différentes, elles en souffrent, car certains des nœuds de calcul peuvent devenir gravement chargés, tandis que d'autres restent inactifs. RRA est généralement utilisé dans les serveurs Web où les demandes HTTP sont généralement de même nature et donc distribuées de manière égale.

(b) Algorithme Randomisé (Randomized Algorithm)

Sans aucune information sur la charge actuelle ou précédente du nœud. Les nœuds de calcul sont sélectionnés de manière aléatoire à la suite de nombres aléatoires générés sur la base d'une distribution statistique (S. El-Zoghdy & Ghoniemy, 2014a; Rajguru & Apte, 2012).

(c) Algorithme du gestionnaire central (Central Manager Algorithm)

À chaque étape de l'algorithme CMA[16], le nœud central (gestionnaire ou maître) sélectionne le nœud esclave ou travailleur auquel une tâche doit être affectée. Le nœud esclave ayant la charge de travail la plus légère est sélectionné. Le nœud central conserve l'indice de charge de tous les nœuds esclaves qui lui sont connectés. Chaque fois que la charge est modifiée, un message est envoyé par les nœuds esclaves au nœud central (S. Sharma, Singh, & Sharma, 2008). Le gestionnaire de charge prend des décisions d'équilibrage de la charge en fonction des informations de charge du système, ce qui permet de prendre la meilleure décision possible au moment du processus créé (S. El-Zoghdy & Ghoniemy, 2014b; S. Sharma et al., 2008).

(d) Algorithme de seuil (Threshold Algorithm)

Dans TA[17], les tâches sont assignées immédiatement lors de la création aux nœuds de calculs. Chaque nœud de calculs conserve une copie privée des informations sur la charge de travail du système. La charge d'un nœud de calculs peut être caractérisée par l'un des

[16] CMA: Central Manager Algorithm
[17] TA: Threshold Algorithm

trois niveaux : *sous-chargé*, *moyen* et *surchargé*. Deux paramètres de seuil, *t_under* et *t_upper*, peuvent être utilisés pour décrire ces niveaux.

1. Sous-chargé : charge de travail < t_under,

2. Moyenne : t_under ≤ charge de travail ≤ t_upper,

3. Surchargé : charge de travail > t_upper.

Tous les nœuds du calcul sont considérés comme sous-chargés à l'état initial. Lorsque l'état de charge d'un nœud de calcul dépasse le seuil de charge, il envoie alors des messages concernant le nouvel état de charge à tous les autres nœuds de calcul, en les mettant à jour régulièrement de sorte que l'état actuel de charge du système puisse être connu (Dong & Akl, 2006). Si l'état local n'est pas surchargé, la tâche est allouée localement. Sinon, un nœud de calcul distant sous-chargé est sélectionné, et si aucun nœud de calcul de ce type n'existe, la tâche est également allouée localement (Rajguru & Apte, 2012).

b) **L'approche d'équilibrage de charge dynamique**

Il existe trois problèmes majeurs dans lesquels un équilibrage de charge statique est impossible ou peut mener à un déséquilibre de charge. Les problèmes sont les suivants :

1) La première classe comprend les problèmes dans lesquels toutes les tâches sont disponibles au début du calcul, mais le temps requis par chaque tâche est différent.
2) La deuxième classe comprend les problèmes dans lesquels les tâches sont disponibles au début, mais au fur et à mesure que le calcul avance, le temps requis par chaque tâche change.
3) La troisième classe comprend les problèmes dans lesquels les tâches ne sont pas disponibles au début, mais sont générées dynamiquement.

En équilibrage de charge statique, trop d'informations sur la charge du système sont nécessaires avant l'exécution, qui ne soit pas possible à chaque fois. L'équilibrage de charge dynamique a donc été développé pour répondre à ces contraintes. L'équilibrage de charge dynamique prend des décisions plus informatives en la matière lors de l'exécution, à l'aide des informations d'état d'exécution (Salehi, Deldari, & Dorri, 2009). Dans les algorithmes d'équilibrage de charge dynamique, la charge de travail est répartie entre les nœuds de calcul au moment de l'exécution. Ces algorithmes surveillent les modifications de la charge de travail du système et redistribuent les tâches en conséquence.

Avantages

1. L'équilibrage de charge dynamique fonctionne bien pour les systèmes hétérogènes.
2. La tâche peut être redistribuée à n'importe quel nœud pendant l'exécution, les problèmes de surcharge et de chargement étant ainsi réduits au minimum.
3. Cela fonctionne bien pour les tâches ayant un temps d'exécution différent.
4. Le système n'a pas besoin de connaître le comportement au moment de l'exécution des applications avant l'exécution.

Les inconvénients

1. La communication est très élevée et devient plus importante lorsque le nombre des nœuds augmente.
2. Les algorithmes d'équilibrage de charge dynamique sont complexes et par conséquent difficiles à mettre en œuvre.
3. Les frais généraux du système augmentent, car ils sont préemptifs.

(a) Les catégories d'équilibrage de charge dynamique

En général, les algorithmes d'équilibrage de charge dynamique peuvent être classés dans les grandes catégories suivantes :

1) Centralisé ou décentralisé

Dans l'approche *centralisée*, un seul nœud joue le rôle de contrôleur principal ou central (maître). Le nœud principal contient l'ensemble des tâches à exécuter. Le nœud principal a une vue globale sur l'état du système (les informations de charge de tous les nœuds qui lui sont connectés) et décide comment attribuer des tâches à chacun des nœuds esclaves. Le reste des nœuds agissent comme des esclaves ou travailleurs (G. Sharma, 2013). Les tâches sont envoyées aux nœuds esclaves pour l'exécution. Il présente un avantage de facilité de mise en œuvre, mais souffre d'un seul point d'échec. Dans l'approche *décentralisée*, tous les nœuds du système sont impliqués dans la décision d'équilibrage de charge. Les informations d'état sont réparties entre tous les nœuds et les nœuds sont responsables de la gestion de leurs propres ressources ou de l'attribution de tâches aux autres nœuds résidant dans leurs files d'attente. Il est plus évolutif et offre ***une meilleure tolérance aux pannes***.

2) Coopérative ou non coopérative

Si un algorithme d'équilibrage de charge distribué est adopté, le prochain problème à prendre en compte est de savoir si les nœuds impliqués dans le processus de distribution

des tâches travaillent en coopération ou indépendamment (de manière non coopérative). Dans *l'approche non coopérative*, chaque nœud est indépendant, à une autonomie par rapport à son propre ordonnancement des ressources, ce qui signifie que les décisions sont prises indépendamment du reste du système (S. El-Zoghdy & Ghoniemy, 2014b). Par conséquent, le nœud peut migrer ou allouer des tâches en fonction des performances locales. Dans *l'approche coopérative*, chaque nœud décideur d'équilibrage de charge a la responsabilité d'exécuter sa propre partie de la tâche de distribution, mais tous les nœuds décideurs travaillent dans le but d'atteindre un objectif commun à l'ensemble du système.

3) Adaptatif ou non adaptatif

L'équilibrage de charge dynamique est dit adaptatif si les décisions prévues prennent en compte les performances passées et actuelles du système et sont affectées par les décisions antérieures ou des changements dans l'environnement. Il est dit non adaptatif si les paramètres utilisés dans l'équilibrage de charge restent les mêmes, quel que soit le comportement passé du système.

4) Émetteur / Récepteur / symétrique

La répartition des tâches dans l'équilibrage de charge dynamique peut être déclenchée par l'émetteur, le récepteur ou symétrique. Dans la stratégie déclenchée par l'émetteur, les nœuds encombrés tentent de transférer les tâches vers des nœuds sous-chargés. Dans la stratégie déclenchée par le récepteur, les nœuds sous-chargés demandent que des tâches leur soient envoyées par les nœuds surchargés. Dans l'approche symétrique, les nœuds sous-chargés et surchargés peuvent lancer des transferts de charge.

(b) Les algorithmes d'équilibrage de charge dynamique

En raison de son impact important sur les performances des systèmes distribués, les algorithmes DLB deviennent un domaine de recherche attrayant pour des nombreux chercheurs. Un grand nombre d'algorithmes DLB ont été développés. Certains de ces algorithmes sont discutés ci-dessous.

1) Algorithme DLB aléatoire (Random DLB Algorithm)

Dans cet algorithme, si un déséquilibre de charge est détecté par le gestionnaire d'équilibrage de charge, le gestionnaire sélectionne de manière aléatoire un nœud de calcul pour recevoir certaines tâches parmi celles surchargées. Il ne vérifie pas les informations d'état de la charge de travail du nœud de calcul sélectionné avant d'envoyer les tâches. À cet effet, cet algorithme ne conserve aucune information de charge de travail locale ni

n'envoie aucune information de charge de travail à d'autres nœuds de calcul. De plus, il est simple à concevoir et à mettre en œuvre. Mais cela entraîne des coûts de communication considérables en raison de la sélection aléatoire de nœuds de calcul sous-chargés. De plus, à la suite de la sélection aléatoire de nœuds de calcul sous-chargés, cet algorithme peut conduire à des cas de déséquilibre. Par conséquent, une dégradation des performances du système (Diekmann, Frommer, & Monien, 1999; S. F. El-Zoghdy, 2012).

2) L'algorithme « nearest-neighbor »

Avec l'algorithme du voisin le plus proche, chaque gestionnaire d'équilibrage de charge considère que seuls ses nœuds de calcul voisins immédiats effectuent des opérations d'équilibrage de charge. Un gestionnaire d'équilibrage de charge prend la décision d'équilibrage en fonction de la charge dont il dispose et des informations de charge transmises à ses voisins immédiats. En échangeant la charge successivement sur les nœuds de calcul voisins, le système atteint un état de charge global équilibré. L'algorithme du voisin le plus proche est principalement divisé en deux catégories, méthode de diffusion et méthode d'échange de dimensions. Grâce à cette méthode de diffusion, un nœud de calcul surchargé ou sous-chargé équilibre sa charge simultanément avec tous ses voisins les plus proches à la fois, tandis qu'en mode d'échange de dimensions, un nœud de calcul équilibre sa charge successivement avec son voisin un à la fois (JavedHussain & DurgeshKumar, 2013).

3) Algorithme probabiliste

Dans cet algorithme, chaque nœud de calcul conserve un vecteur de charge contient la charge d'un sous-ensemble de nœuds de calcul. La première moitié du vecteur de charge contient également la charge de travail locale, qui est envoyée périodiquement à un nœud sélectionné aléatoirement. Les informations sur la charge de travail du système sont mises à jour de cette manière afin de minimiser le nombre des messages échanger dans le système et le temps système de communication. Cet algorithme n'est pas évolutif, c'est-à-dire que son extensibilité est faible (Ali & Khan, 2012)

4) Algorithmes de seuil et l'algorithme moins (Threshold Algorithms and least Algorithms)

Ces algorithmes utilisent une connaissance partielle des informations sur la charge de travail du système obtenue par les échanges de messages. Un nœud de calcul est sélectionné de manière aléatoire pour accepter les tâches migrées. Si la charge de travail du nœud sélectionné est inférieure à une certaine limite (la charge seuil), les tâches migrées sont

acceptées pour y être calculées. Sinon, l'interrogation est répétée avec un autre nœud de calcul pour trouver celui qui est approprié. Après un nombre maximal de tentatives si aucun destinataire approprié n'a été signalé, les tâches sont exécutées localement (Ali & Khan, 2012). *LEAST* est une instance de *THRESHOLD*, après l'interrogation, la machine là moins chargée est choisie pour recevoir la charge migrée (Bernard, Stève, & Simatic, 1993). *THRESHOLD* et *LEAST* ont de bonnes performances et sont de nature simple. De plus, les informations utilisées sur la charge du système sont à jour.

5) Information centralisée et décision centralisée

Dans ces algorithmes, toutes les informations sur la charge de travail du système sont stockées sur un seul nœud (le nœud maître) et les décisions d'équilibrage de la charge sont également prises par ce nœud. Chaque nœud de calcul du système met à jour le nœud maître en fonction de ses informations de charge de travail, soit périodiquement, soit en cas de modification de sa charge de travail. Lorsqu'un nœud sur chargé souhaite migrer une partie de sa charge de travail, il demande au nœud maître un nœud sous-chargé. Chaque nœud du système indique à la machine serveur si un nœud légèrement chargé est disponible ou non (Ali & Khan, 2012). Ce type d'algorithme donne des résultats de performance très efficaces, mais souffre d'un problème très grave: si le nœud maître tombe en panne ou ne fonctionne pas correctement, l'ensemble du processus sera arrêté, c'est-à-dire un point de défaillance unique. Conséquence directe du problème du point de défaillance unique.

6) Information centralisée et décision distribuée

Globalement, la collecte des informations sur la charge de travail du système est centralisée (au niveau d'un nœud maître), mais la décision est répartie (Ali & Khan, 2012). Les informations sur la charge de travail du système des nœuds de calcul sont diffusées par le nœud maître. En utilisant ces informations, un nœud de calcul surchargé trouve ceux qui sont sous-chargés à partir de son vecteur de charge sans passer par le nœud maître. Cet algorithme est très efficace en raison de la moindre inclusion d'informations de message et de sa robustesse, car le système reste actif même lorsque le nœud maître est en état de récupération. Cette classe d'algorithmes DLB collecte une grande quantité d'informations sur la charge de travail du système, mais elle n'est peut-être pas à jour.

7) Informations distribuées et décision distribuée

Dans ce type d'algorithmes DLB, chaque nœud de calcul met à jour les autres nœuds de calcul en fonction de ses informations d'état de charge de travail, à l'aide de l'une des stratégies suivantes :

a. *Une diffusion périodique :* chaque nœud de calcul diffuse ses informations d'état de charge de travail uniquement lorsque l'état du nœud change en raison de l'arrivée ou du départ d'une tâche.
b. *Échange à la demande :* un nœud de calcul diffuse un message de demande d'information d'état lorsque son état passe de l'état normal à la région sous-chargée ou surchargé. Ensuite, les autres nœuds de calcul envoient leurs informations sur l'état de la charge de travail actuelle.

Chaque nœud de calcul conserve un vecteur d'informations de charge de travail global sur lequel il prend les décisions d'équilibrage de charge. En cas de mise à jour périodique, la durée de la mise à jour des informations sur la charge de travail a un impact important sur les performances du système. En effet, si elle est trop courte, la décision d'équilibrage de la charge sera prise en fonction des informations de charge de travail à jour. Mais d'autre part, cela augmentera la charge de communication du système, ce qui pourrait dégrader les performances de ce type d'algorithmes DLB (Ali & Khan, 2012; Bernard et al., 1993).

2.8 Travaux sur la tolérance aux pannes et l'équilibrage de charge

Plusieurs travaux de recherche sont focalisés sur l'équilibrage de charge sous contrainte, les algorithmes de découverte des ressources basés sur le cluster, le Grid, les DGVCS, et le Cloud.

Nous présentons les travaux réalisés pour résoudre : le problème d'équilibrage de charge et la tolérance aux pannes dans les différents systèmes distribués.

Dans (Anderson & McLeod, 2007), les auteurs décrivent les règles de planification locales utilisées dans le client BOINC. Ces stratégies peuvent fonctionner dans un large éventail de scénarios de planification réels. Il existe trois aspects clés pour ces politiques : 1) la notion de dette; 2) l'utilisation de la planification des délais; et 3) une attention particulière à l'estimation de l'achèvement des travaux. Les stratégies de planification locales ne reflètent actuellement les contraintes de mémoire qu'après coup. Aussi, on peut noter que les délais ne sont pas respectés.

Dans (Dabrowski & Hunt, 2009), les auteurs ont introduit une approche pour l'utilisation de chaînes de Markov discrètes, homogènes et fragmentées, afin de fournir une simulation rapide et potentiellement évolutive de systèmes de grille à grande échelle. Leur approche est utilisée pour modéliser des systèmes de grille dynamiques à grande échelle. En outre, un modèle de chaîne de Markov d'un système de grille est représenté sous une forme réduite et compacte. Ce modèle peut être ensuite perturbé afin de produire des chemins d'exécution alternatifs et d'identifier des scénarios dans lesquels les performances du système risquent de se dégrader ou d'apparaître des comportements anormaux. La génération rapide de ces scénarios permet de prédire comment un système plus grand réagira aux pannes ou aux conditions de stress élevé. Cependant, l'effort de calcul augmente proportionnellement au nombre de chemins modélisés. Ce coût est bien inférieur au coût de la simulation détaillée ou des bancs d'essai. De plus, les coûts ne sont pas affectés par la taille du système modélisé, exprimés en termes de charge de travail et de nombre de ressources de calcul, et sont adaptables aux systèmes non homogènes en fonction du temps. Ce dernier papier cité fournit des exemples détaillés de l'application de cette approche.

Dans (Savio, 2009), l'auteur a défini trois algorithmes en ligne permettant d'équilibrer la charge entre deux critères indépendants avec une réallocation d'objet. L'auteur a défini l'algorithme d'équilibrage des limites en ligne sur la base de deux critères indépendants, avec réplication et réallocation. Cet algorithme reflète trois résultats en ligne permettant d'équilibrer les espaces de charge et de stockage sur des serveurs homogènes. La charge fait référence à un paramètre tandis que l'équilibrage de la charge fait référence à un problème classique.

Les auteurs de (Mokadem et al., 2012) se sont concentrés sur la découverte de ressources. Ils ont utilisé une solution DHT hiérarchique à passerelle unique (SG-HDHT). Cette dernière traite à la fois de la réduction des coûts de recherche et de la gestion du désabonnement. Tout en minimisant les frais généraux supplémentaires pour le système. Il prend également en compte la localité de contenu/chemin des organisations dans Grids. Les méthodes proposées par les auteurs reflétant les performances positives des requêtes de recherche, notamment lorsqu'il existe une augmentation du nombre de messages de découverte de ressources simultanément. Cela concerne à la fois les requêtes intra et inter-VO. Ils ont également montré une réduction significative du trafic réseau et offre d'importantes économies de maintenance DHT, en particulier lorsque des nœuds rejoignent/quittent fréquemment le système.

Le problème pour la prédiction de la disponibilité à long terme dans les systèmes de calculs volontaires a été présenté en (Lázaro et al., 2012). Dans leur travail, ils l'ont appliqué à un mécanisme de déploiement de service sensible à la disponibilité pour des systèmes de calculs volontaires. Ce mécanisme a été inspiré de l'analyse de la disponibilité de l'hôte dans des traces système réelles (projet informatique bénévole SETI@home). Ils se sont concentrés sur le problème de la prévision si un groupe de ressources serait disponible en permanence pendant une période relativement longue.

Une Architecture overlay a été présentée par (Pitoura et al., 2012) pour les systèmes de données DHT à grande échelle, elle aborde trois problèmes dans les réseaux de données P2P structurés: le traitement efficace des requêtes par intervalle, l'équilibrage de la charge et la tolérance aux pannes. Saturne traite efficacement ces problèmes par le biais du nouveau multiple ring. Pour garantir un traitement rapide des requêtes de plage, ils utilisent une nouvelle fonction de hachage préservant les ordres. La réplication verticale et horizontale constitue la base sur laquelle leurs mécanismes sont développés, garantissant ainsi l'équilibrage de la charge des requêtes et la tolérance aux pannes.

(Tse, 2013) a étudié dans un système de serveurs de fichiers homogènes distribués situés dans un cluster, la nature de la réplication d'objet dans le contexte de l'équilibrage de charge à deux critères. L'auteur a proposé en ligne des solutions approximatives permettant d'équilibrer la charge et les espaces de stockage requis lors des placements, avec la possibilité de deux répliques, voire aucune. La limite supérieure de la charge est considérablement réduite, sans sacrifier l'espace de stockage, par rapport au meilleur algorithme de placement simple existant. L'auteur a donné un algorithme en ligne amortissant la complexité temporelle ($logM$), et la complexité temporelle du cas le plus défavorable ($MlogM$), où M est le nombre de serveurs. La faible complexité temporelle amortie est garantie si la complexité temporelle dans le pire des cas est réellement atteinte.

(Saini & Prakash, 2013) ont proposé une nouvelle architecture basée sur l'architecture middleware de messagerie basée sur XMPP afin d'améliorer la communication entre les clouds volontaires et les autres infrastructures de clouds commerciaux. Cette architecture offre une large gamme de services, tels que la communication bidirectionnelle, éliminant les longues interrogations, l'interopérabilité améliorée et la disponibilité. Les auteurs se sont concentrés sur un protocole de communication via un middleware orienté message et non sur la sécurité de leur plate-forme hybride.

(Javadi, Matawie, & Anderson, 2013) se sont concentrés sur la modélisation statistique de la disponibilité des ressources volontaires, qui présente une configuration non aléatoire

dans leur temps de disponibilité. Où ils les ont divisés en un sous-ensemble d'hôtes dont la disponibilité dépend de la dépendance à court/long terme. En outre, ils ont appliqué trois modèles statistiques, à savoir MMPP, MAP et MWM, sur des traces réelles du projet SETI@home avec plus de 230000 hôtes.

(Ghafarian et al., 2013) ont suggéré un modèle analytique pour l'équilibrage de la charge dans des systèmes de calculs peer-to-peer basés sur des volontaires. Ils ont considéré que les demandes arrivant dans le système se présentaient sous la forme d'un paquet de tâches (BoT), c'est-à-dire l'application contenant certaines des tâches parallèles indépendantes, qui s'exécuteraient sur un seul nœud, et chaque demande aurait des contraintes de qualité de service en termes de vitesse du processeur, besoins en RAM ou en espace disque. L'algorithme de découverte de ressources proposé est composé de deux phases. Dans la première phase, il prend en compte la fonctionnalité d'équilibrage de charge dans le système. Dans la deuxième phase, la ressource ayant le temps système de communication minimal est sélectionnée.

Les problèmes d'équilibrage de charge, de tolérance aux pannes et de fiabilité pour l'environnement de mobile grids ont été traités par (Behera & Tripathy, 2014). Les auteurs ont proposé deux algorithmes : l'un pour l'équilibrage de charge décentralisé et l'autre pour la tolérance aux pannes. Les algorithmes proposés ont amélioré la complexité temporelle et l'efficacité par rapport aux travaux existants. Le papier propose également une méthode de modélisation de la fiabilité du réseau mobile. Les méthodes proposées peuvent être utilisées pour rechercher la disponibilité et le délai moyen d'échec avec d'autres améliorations.

Le travail présenté dans (Bala & Chana, 2015) formule un effort centré sur la manière de faciliter l'exécution de tâches simultanées dans des problèmes de flux de travail scientifiques. Pour concevoir une approche de planification autonome à tolérance de pannes pour les applications de flux de travaux scientifiques, les auteurs ont proposé une heuristique hybride pour planifier efficacement les flux de travaux scientifiques et, deuxièmement, utiliser une approche de migration de machine virtuelle comme technique de tolérance aux pannes pour migrer automatiquement la machine virtuelle en cas d'échec en raison de la surutilisation des ressources. En outre, ils valident cette approche à l'aide de paramètres d'évaluation des performances utilisant CloudSim et WorkflowSim. Les résultats de la simulation démontrent l'efficacité de l'approche proposée pour améliorer les performances des workflows scientifiques en réduisant sensiblement le temps moyen total d'exécution, le temps d'écart type et la durée d'exécution.

Une approche de calculs décentralisée pour attribuer et programmer des tâches sur un réseau à distribution massive Grids a été introduite par (Banerjee & Hecker, 2017), et est basée sur des systèmes multi-agents. Pour répondre à l'évolution des besoins en ressources, ils proposent un algorithme pour créer et dissocier de manière dynamique les clusters du travail. Ce travail a été comparé à un algorithme de planification premier entré, premier sorti (FIFO[18]), sur le temps nécessaire pour vider la file d'attente, le temps d'attente moyen et l'utilisation de CPU. Cette approche de calculs décentralisée est trouvée prometteuse pour des scénarios de traitement massivement distribués tels que SETI@home et Google MapReduce.

Dans (Hemam & Hioual, 2017), les auteurs traitent les problèmes d'équilibrage de charge en fonction des besoins des utilisateurs (l'optimisation du coût et de la fiabilité), dans la plate-forme hybride P2P et Cloud. Ils proposent un modèle de système cloud P2P combinant des ressources fiables à coût réduit et des ressources non fiables à coût zéro. En outre, afin de trouver la solution optimale, ils appliquent l'algorithme de Pareto font pour sélectionner les ressources non dominées. Après ça, ils appliquent la technique (TOPSIS) pour obtenir la solution idéale, adaptée aux contraintes de l'utilisateur.

Dans (Mok, Bajpai, Dhamdhere, & Claffy, 2018), les auteurs appliquent des techniques pour équilibrer la charge du volume extraordinaire de requêtes Web et du trafic provenant d'utilisateurs de YouTube. Après cela, ils attribuent des requêtes YouTube à des caches de contenu vidéo Google spécifiques, notamment les liens d'interconnexion entre les fournisseurs d'accès et Google. Les mesures utilisées ont été collectées à partir de sondes SamKnows hébergées par des clients large bande couvrant un important fournisseur de services Internet aux États-Unis et trois fournisseurs de services Internet en Europe entre la mi-2016 et la mi-2017. Les deux causes possibles de l'utilisation différente de liens inter-domaines ont été examinées (l'emplacement géographique et l'heure de la journée). Une comparaison entre les noms d'hôte du cache vidéo YouTube et les adresses IP observées par les sondes a également été effectuée. Enfin, ils ont prouvé que la sélection du cache vidéo avait peu d'impact sur la sélection BGP des liens inter-domaines.

Afin d'affiner l'analyse des différents travaux de recherche relatifs à l'équilibrage de la charge entre les ressources, et l'utilisation de la tolérance aux pannes nous proposons un

[18] FIFO: First-In First-Out

tableau comparatif (voir Tableau 2.1), organisée selon les caractéristiques principales suivantes :

La plate-forme, la technique utilisée pour détecter la panne, le composant défectueux, la prise en charge de l'équilibrage de charge sous les exigences des utilisateurs ou non, la disponibilité, la fiabilité, et les techniques de récupérations utilisées.

D'après le Tableau 2.1, on peut constater que les travaux de [(Savio, 2009), (Tse, 2013)] prennent en considération l'équilibrage de charge de façon générale et sans tenir en compte les exigences de l'utilisateur, ainsi que la tolérance aux pannes. Dans les travaux de [(Mokadem et al., 2012); (Pitoura et al., 2012); (Behera & Tripathy, 2014)], la tolérance aux pannes est prise en considération, et la technique de recouvrement utilisée dans ce cas est réactive, par contre ils ne prennent pas en considération les exigences des utilisateurs lors de l'équilibrage de charge. Les auteurs des travaux (Bala & Chana, 2015) ont traité seulement la tolérance aux pannes alors que le problème de l'équilibrage de charge n'a pas été pris en considération. Dans le contexte des systèmes de calcul volontaire, on peut remarquer que l'un des travaux (Lázaro et al., 2012) a pris en considération la disponibilité et la fiabilité des ressources sans tenir en compte du problème de l'équilibrage de charge contrairement aux travaux de (Ghafarian et al., 2013). Dans (Hemam & Hioual, 2017), les auteurs ont traité le problème de l'équilibrage de charge en prenant en considération les exigences des utilisateurs, et la technique de recouvrement utilisée est réactive.

A la différence des travaux précédemment présentés, notre contribution traite le problème de la tolérance aux pannes en utilisant une technique hybride (proactif et réactif). Ainsi, notre proposition permet de : (1) prévenir l'apparition future des pannes d'un côté, (2) équilibrer la charge entre les différentes ressources du système avec la prise en considération des préférences des utilisateurs d'un autre côté.

Sachant cela, certaines des approches citées dans ce tableau sont basées sur des techniques de détection de pannes. De plus, nous notons que la plupart des recherches concernant les systèmes de calculs volontaires sont davantage axées sur la méthode réactive que sur les méthodes proactives pour la détection de la défaillance, ainsi que l'équilibrage de charge basé sur les performances de system et non sur les exigences des utilisateurs.

Source	Plate-forme	Techniques utilisées	Composant défaillant	Mode proactif ou réactif	Adapter le passage à échelle	Prise en charge de l'équilibrage de charge		La disponibilité	La fiabilité	Technique de récupération utilisée
						Sans les exigences d'utilisateur	Sous les exigences d'utilisateur			
(Savio, 2009)	P2P Client/ Serveur	Non	Non	Non	✓	✓	✗	✗	✓	Réplication Réallocation
(Mokadem et al., 2012)	Grid DHT	Heartbeat Monitor (HBM)	Échec de l'hôte	Réactif	✓	✓	✗	✗	✗	Réplication
(Pitoura et al., 2012)	Grid DHT	Heartbeat Monitor (HBM)	Échec de l'hôte	Réactif	✓	✓	✗	✓	✗	Réplication
(Lázaro et al., 2012)	Système de Calcul volontaires	Heartbeat Monitor (HBM)	Échec de l'hôte	Proactif	✓	✗	✗	✓	✓	Réplication
(Tse, 2013)	Cluster	Non	Non	Non	✓	✓	✗	✗	✗	Réplication
(Ghafarian et al., 2013)	Système de Calcul volontaires	Non	Non	Non	✓	✓	✓	✗	✗	Réplication
(Behera & Tripathy, 2014)	Mobile Grid Computing Systems	Heartbeat Monitor (HBM)	Échec de l'hôte	Réactif	✓	✓	✗	✗	✓	Réplication
(Bala & Chana, 2015)	Cloud	Heartbeat Monitor (HBM)	Machines virtuelles	Réactif	✓	✗	✗	✓	✗	Check pointing
(Hemam & Hioual, 2017)	Hybrid platform P2P & Cloud	Heartbeat Monitor (HBM)	Échec de l'hôte	Réactif	✓	✓	✓	✓	✗	Resubmit job

Tableau 2.1 Tableau de comparaison entre quelques travaux récents.

2.9 Conclusion

La tolérance aux pannes est un sujet important dans les systèmes distribués, car elle est considérée comme un thème important dans la conception des systèmes distribués. Fondamentalement. La tolérance aux pannes correspond à la capacité d'un système à fonctionner en cas de défaillance. Afin de réduire l'impact des pannes sur un système, de nombreuses techniques de tolérance aux pannes sont disponibles et peuvent être utilisées au niveau du service, de l'application ou du matériel. Dans ce chapitre une description de l'équilibrage de charge dans les systèmes distribués est bien détaillée et discutée, avec les approches et les algorithmes trouvés dans la littérature. Dans le chapitre suivant, nous décrirons notre contribution pour la tolérance aux pannes.

Chapitre 3 CONTRIBUTION POUR LA TOLERANCE AUX PANNES ET L'EQUILIBRAGE DE CHARGE

3.1 Introduction

Dans les systèmes de calculs volontaires, les ressources sont fournies par les hôtes. Ces systèmes sont devenus plus puissants et plus attrayants, car ils fournissent une puissance de calculs énorme. Cependant, répondre aux besoins des utilisateurs et aux performances du système dans ce type de paradigme est un défi crucial.

Nous détaillons dans ce chapitre l'architecture de notre approche basée sur un système de calcul volontaire. L'architecture doit être structurée de telle sorte que la disponibilité, la transparence, l'équilibrage de charge, la tolérance aux pannes, et le passage à l'échelle soient garantis.

3.2 Contribution

Dans notre travail, nous proposons une nouvelle architecture basée sur un système de calcul volontaire, dans lequel les ressources sont fournies par les participants publics. Dans cet environnement, nous abordons deux problèmes importants : l'équilibrage de charge entre les ressources volontaires et l'optimisation des deux critères : le temps de réponse et la disponibilité. Ces deux critères (le temps de réponse et la disponibilité) sont souvent contradictoires et ont un impact négatif sur la charge entre les ressources, c'est-à-dire que si les utilisateurs doivent maximiser le critère de disponibilité, les demandes sont acheminées vers les ressources volontaires les plus élevées disponibles, ce qui provoque une augmentation de la charge des ressources et par conséquent le temps de réponse. D'autre part, si les utilisateurs doivent minimiser le temps de réponse, les demandes sont alors acheminées vers les ressources volontaires caractérisées par une faible disponibilité (car ils sont sous-chargés), mais il y a un risque que les ressources tombent en pannes et les tâches soit renvoyées vers d'autre ressource, ce qui provoque l'augmentation de temps de réponse. Dans ces cas, certaines ressources volontaires sont surchargées et d'autres le sont insuffisamment. Pour régler ce problème, nous proposons une architecture composée de trois composants globaux :

1. ***Les sous-groupes volontaires*** : contenant des ressources de volontaires peu fiables.

2. ***Les utilisateurs :*** qui envoient leurs demandes aux couches de commutation en indiquant leurs besoins en matière de préférences.
3. ***La couche de commutation :*** composée d'un ensemble des switchers balancers (super-nœuds ou équilibreurs switchers) connectés entre eux via une connexion fiable.

Les switchers balancers jouent un rôle important dans ce système, car :

- Ils permettent d'équilibrer d'une part la charge de manière distribuée entre eux et d'autre part entre les ressources volontaires de manière centralisée de leur sous-groupes volontaires.
- Ils estiment la probabilité de défaillance de chaque ressource volontaire durant chaque heure afin de l'utiliser ultérieurement.
- Ils sélectionnent également la ressource volontaire la plus appropriée en fonction des besoins de l'utilisateur (l'optimisation du temps de réponse et de la disponibilité).

Pour ce faire, notre approche passe par deux étapes :

- Dans la première étape, nous sélectionnons un sous-groupes volontaire ***actif*** et ***sous-chargé***, puis les ressources volontaires sous-chargées sont sélectionnées afin de satisfaire les performances du système.
- Dans la deuxième étape, nous proposons un algorithme qui sélectionne la ressource volontaire approximative considérée comme la meilleure alternative des ressources volontaires en fonction des besoins de l'utilisateur (optimisation du temps de réponse et de la disponibilité).

De plus, nous utilisons le modèle de chaîne de Markov à processus stochastique pour estimer la probabilité d'échec de chaque ressource volontaire.

Dans la section suivante nous définissons l'architecture et le modèle du système proposé.

3.3 Architecture et modèle proposé

Dans cette section, nous présentons tout d'abord dans la sous-section suivante (3.3.1) l'architecture globale du système pour clarifier cette proposition. De plus, nous présentons notre modèle de système dans la section 3.3.2.

3.3.1 Architecture du système

Comme le montre la Figure 3.1, l'architecture du système décrite dans notre contribution contient deux couches : *couche d'utilisateur* et *couche du système*.

1) La couche du système est divisée en deux sous-couches :
   - Couche des switchers (couche de commutation) : est un ensemble de switchers balancers nommés aussi des super-nœuds, qui sont des nœuds fiables formant un système de quorum.
   - Couche ressources volontaires : des ensembles de ressources volontaires peu fiables, c'est-à-dire des pairs instables.
2) Couche utilisateur : contient les clients, qu'ils envoient leurs demandes au switcher balancer et obtiennent des réponses.

3.3.2 Modèle de système

a) Couche ressources volontaires

Nous supposons que le système est composé d'un ensemble de M ressources volontaires désignées par VR_i où $i \in [1..M]$. Ces ressources sont divisées en sous-ensembles, en tenant compte leur emplacement géographique et des services qui seront offerts. Chaque sous-ensemble forme un sous-groupe volontaire.

Nous considérons également que le nombre de sous-groupes volontaires dans le système est K, le sous-groupe volontaire (i) est alors noté $VCL_{i[1..K]}$.

$S_{VR}^{VCL_i}$ est un ensemble de ressources volontaires sur le sous-groupe volontaire VCL_i. Les conditions suivantes doivent être remplies :

$$S_{VR}^{VCL_1} \cap S_{VR}^{VCL_2} \cap \ldots \cap S_{VR}^{VCL_k} = \emptyset$$

$$S_{VR}^{VCL_1} \cup S_{VR}^{VCL_2} \cup \ldots \cup S_{VR}^{VCL_k} = SVCL$$

Nous considérons LVR_i comme le temps de réponse nécessaire pour exécuter la demande de l'utilisateur par la ressource volontaire VR_i.

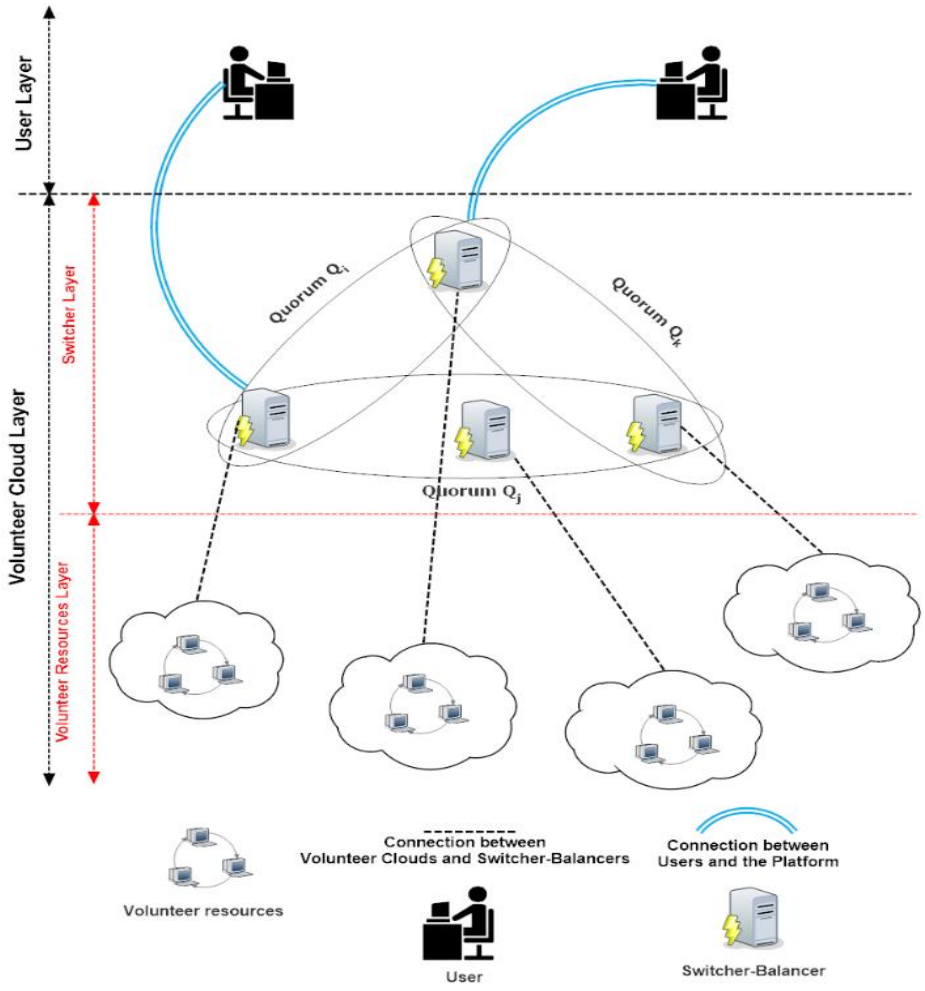

Figure 3.1 Architecture du système proposé

b) **Couche des switchers**

Pour de nombreuses applications dans les systèmes distribués, des systèmes de quorum, qui sont une collection de sous-ensembles de nœuds voire Figure 3.2, ont été utilisés pour réduire le nombre de messages échangé dans le système, y compris l'exclusion mutuelle, la réplication de données et le partage d'informations. Afin de minimiser le nombre de messages, nous introduisons un ensemble de switchers balancers nommé $Sets_{SB}$, ainsi qu'un ensemble $Q = \{Q_1, Q_2, \dots, Q_t\}$, où $Q_i \subseteq Sets_{SB}$.

Un système de quorum Q_i est défini sur l'ensemble $Sets_{SB}$ si et seulement si

Q possède la propriété d'intersection suivante: $\forall\, i, j \in \{1 \dots t\}, Q_i \cap Q_j \neq \emptyset$

Comme illustré à la Figure 3.2.

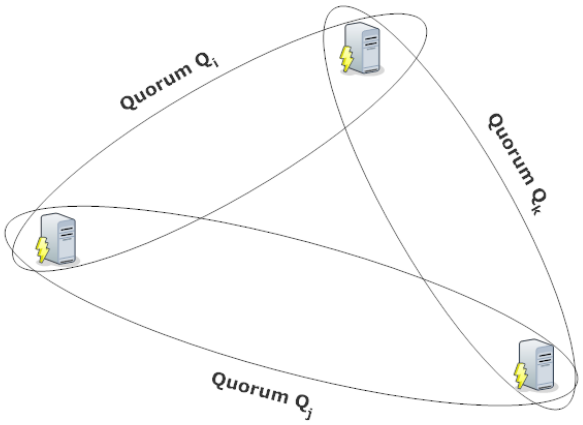

Figure 3.2 Système de Quorum

Pour créer le système de quorum, nous définissons une séquence numérique comme suit :

$$n + (n-1) + (n-2) + \cdots + \big(n-(x-1)\big) = Tot\_SB \ldots. (1)$$

Comme (1) est une suite numérique, nous avons la somme S :

$$Tot\_SB = \frac{x}{2}(2n - x + 1), n \geq {}^{x}/_{2}$$

Où :

Tot_SB : est le nombre de switchers balancers dans le système ;

n: est le nombre de switchers balancers dans chaque quorum ;

x: est le nombre de quorums ;

3.4 Fonctionnement des algorithmes proposés

Comme mentionné dans (Ghomi, Rahmani, & Qader, 2017), le problème de l'équilibrage de charge est classé en deux principaux groupes (statique et dynamique). Ces approches dépendent de la manière dont un algorithme d'équilibrage de charge est basé pour prendre sa décision, c'est-à-dire qu'il est basé sur l'état actuel du système ou non.

Dans l'approche dynamique, les algorithmes d'équilibrage de charge ne sont basés que sur l'état actuel du système. Lors de l'exécution, nous faisons varier de manière dynamique la charge des ressources surchargées vers les ressources sous-chargées (Ledmi et al., 2018). Contrairement à l'approche statique, dans laquelle l'équilibrage de charge est réalisé en fournissant des informations préalables sur le système, une fois que la demande est allouée à une ressource, elle ne peut pas être transférée ou migrée vers une autre ressource.

Presque tous les algorithmes dynamiques doivent suivre quatre étapes (Ghomi et al., 2017):

- ***Surveillance de la charge*** : cette étape permet de surveiller la charge et l'état des ressources.
- ***Synchronisation*** : à cette étape, les informations de charge et d'état sont échangées. (Ressources sous-chargées et surchargées).
- ***Critères de rééquilibrage*** : à cette étape, nous devons calculer la nouvelle charge de travail, puis prendre des décisions d'équilibrage de la charge en fonction de ce nouveau calcul.
- ***Migration de tâches*** : cette étape ne sera exécutée que lorsque le système décidera de transférer une tâche ou un processus.

Dans un système distribué, les algorithmes d'équilibrage de charge dynamique peuvent être classés en deux catégories : ceux *centralisés* et ceux *décentralisés*. De manière centralisée, les demandes sont routées vers le nœud central et toutes les opérations doivent passer par ce nœud. Contrairement à l'algorithme d'équilibrage de charge dynamique décentralisé (S. F. El-Zoghdy & Elnashar, 2015), tous les nœuds doivent interagir entre eux sous une forme coopérative ou non coopérative pour réaliser l'équilibrage de charge.

Dans cette thèse, nous traitons à la fois des algorithmes d'équilibrage de charge dynamique *décentralisés* et *centralisés*, car chaque switcher balancer interagit avec son groupe de quorum de manière décentralisée pour partager des informations et équilibrer la charge dans le système. De plus, il interagit de manière centralisée avec les ressources volontaires appartenant à son sous-groupe volontaire, pour équilibrer la charge entre elles. Ces techniques permettent d'acheminer la demande vers la ressource sous-chargée en prenant en compte les besoins de l'utilisateur (l'optimisation de temps de réponse et de la disponibilité).

3.4.1 Description des algorithmes proposés

Cette section est consacrée à l'explication des algorithmes proposés. Elle est divisée en six sous-sections : l'algorithme d'équilibrage de charge global (algorithme 1), l'algorithme d'équilibrage de charge centralisé (algorithme 2), l'algorithme de sélection (Algorithme 3), l'algorithme d'équilibrage de charge décentralisé (algorithme 4), l'algorithme de calcul de charge moyenne du système (Algorithme 5), et enfin l'algorithme de tolérance de pannes (algorithme 6). Ces six algorithmes doivent être exécutés par chaque switcher balancer.

a) Description de l'algorithme 1 (Algorithme d'équilibrage de charge global)

Cet algorithme doit être exécuté par chaque switcher balancer afin d'équilibrer la charge entre les switchers de manière décentralisée. Ainsi qu'entre les ressources d'un sous-groupe volontaires de manière centralisée.

Le switcher balancer est le composant le plus interactif. Il peut recevoir six types de messages et envoyer trois autres types, comme illustré à la Figure 3.3 dans laquelle :

- 1 représente un message de requête d'un utilisateur afin de le rediriger vers un autre switcher balancer ou de l'exécuter dans l'une de ses ressources volontaires et de renvoyer la réponse.
- 2 peut être un message de demande provenant d'un autre switcher balancer ou un message de quorum provenant aussi d'un autre switcher balancer afin de renvoyer une liste de quorum en réponse.
- 3 est un message de déconnexion. Une ressource volontaire envoie ce type de message pour quitter délibérément le système.
- 4 est un message de fin d'exécution, envoyé par une ressource volontaire pour renvoyer le résultat de la demande de l'utilisateur. Ce message est envoyé depuis un switcher balancer en tant que réponse au message (numéro 7).
- 5 est un événement de dépassement de temps (time out) lorsqu'il n'y a pas de réponse d'une ressource volontaire concernant un message Ping.
- 6 est un message de réponse d'une ressource volontaire à un message Ping envoyé par le switcher balancer en tant que message (numéro 9).
- 7 est message de demande de travail envoyé par l'utilisateur.
- Le numéro 8 est un message de connexion envoyé par une ressource volontaire pour rejoindre le système.
- 9 est message PING.

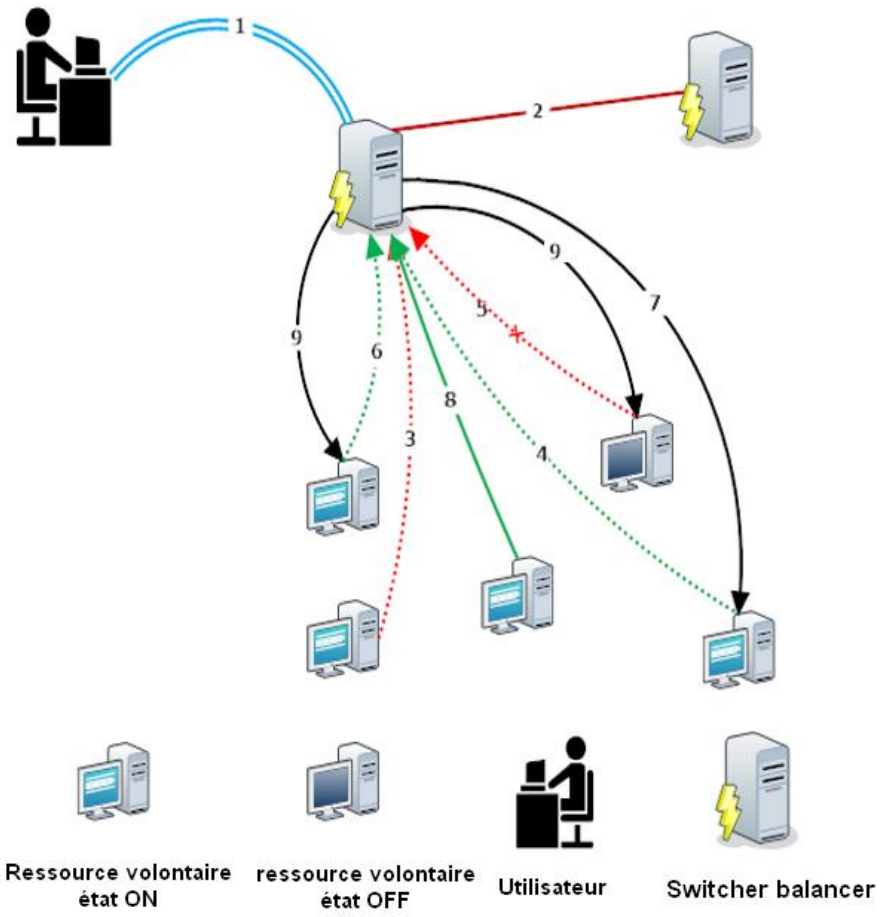

Figure 3.3 L'interaction entre les différents nœuds

Comme indiqué dans l'algorithme 1, le switcher balancer réagit aux messages d'arrivée et les traites en fonction de son type.

Nos algorithmes d'équilibrage de charge sont utilisés dans deux situations :

La première situation en (Algorithme1, ligne 03) est lorsqu'une demande arrive de la part d'un utilisateur, le switcher balancer doit équilibrer la charge entre les switchers balancers (de manière décentralisée). Il appelle l'algorithme 3 dans (Algorithme1, ligne 04).

La deuxième situation en (Algorithme1, ligne 06) est quand une requête utilisateur arrive d'un autre switcher balancer (cela signifie que ce switcher balancer est un switcher balancer actif sous chargé), (voie centralisée), le switcher balancer doit équilibrer la charge entre les ressources volontaires qu'il contrôle. Il appelle (Algorithme 2) dans (Algorithme1, ligne 07), puis met à jour respectivement sa charge ainsi que le tableau de charge (Algorithme1, lignes 08 et 09).

ALGORITHM 1: Global load balancing algorithm

```
Begin
01 while true
02     Arrival Message
03     if (Message = Rq: Request from user) then
04         Algorithm4(Rq)
05     end if
06     if (Message = Rq: Request from another switcher balancer) then
07         Algorithm2(Rq)
08         Update (L_VCL_i ) of this switcher balancer
09         L[i]←L_VCL_i
10     end if
11     if (Message = end of Rq execution) then
12          Remove Rq from the set of requests awaiting execution at this
          volunteer resource
13         Update (L_VR_i )
14         Update (L_VCL_i) of this switcher-balancer
15         L[i] ←L_VCL_i
16     end if
17     if (Message = Reply from (VR_i)) then
18         if (getState(VR_i)=Connected) then
19             P_CC VR_i ← P_CC VR_i +1;
20             StopTime VR_i ;
21         end if
22     end if
23     if Message = Connected (VR_i)
24         if (getState(VR_i)=Connected) then
25             P_CF VR_i ← P_CF VR_i +1;
26             P_FF VR_i ← P_FF VR_i +1;
27             P_FC VR_i ← P_FC VR_i +1;
28         else
29             P_FC VR_i ← P_FC VR_i +1;
30             setState(VR_i,Connected);
31         end if
32     end if
33     if Message=Failure (VR_i) then
34         P_CD VR_i ← P_CD VR_i +1;
35         Algorithm6(VR_i);
36         setState(N_i,Disconnected) ;
37     end if
38     if event = TimeOut (VR_i) then
39         P_CF VR_i ← P_CF VR_i +1;
40         Algorithm6(VR_i);
41         setState(VR_i,DisConnected);
42     end if
43 end while
end algorithm
```

Si une ressource volontaire termine un travail, elle envoie un message de fin d'exécution (algorithme 1, ligne 11) à son switcher balancer afin de supprimer la demande de ce travail

accompli de l'ensemble des demandes en attente d'exécution sur cette ressource volontaire (algorithme 1, ligne 12), puis le switcher balancer met à jour respectivement la charge de cette ressource volontaire, la charge de son sous-groupe volontaire, ainsi que son tableau de charge (L []) (lignes de l'algorithme 1 de 13 à 15).

Dans le cas de réception d'un message de réponse d'une ressource volontaire (algorithme 1, ligne 17), en réponse au message ping (Algorithme PING, ligne 15), (Algorithme 1) vérifie d'abord l'état de cette ressource volontaire, si elle est connectée, c'est-à-dire cette ressource volontaire est toujours connectée, dans ce cas le switcher balancer met à jour l'état de probabilité de transition de cette ressource volontaire (Algorithme 1, ligne 19), puis arrête le délai d'attente lié à cette ressource (ligne 20 de l'algorithme 1).

Dans le cas où le switcher balancer reçoit un message d'une ressource volontaire pour rejoindre le système, il vérifie d'abord l'état de cette ressource volontaire, si elle est connectée (algorithme 1, ligne 24), c'est-à-dire que la ressource volontaire a échoué avant le cycle de vérification. Dans ce cas, le switcher balancer met à jour les valeurs des états de probabilités de transitions. Dans le second cas (l'état de la ressource volontaire est déconnecté), le switcher balancer met à jour uniquement l'état de probabilité de transition et réinitialise l'état de la ressource volontaire à l'état connecter (ligne 30 de l'algorithme 1).

Le cas de message Failure, lorsqu'une ressource volontaire envoie ce type de message pour quitter délibérément le système. Dans ce cas, le switcher balancer met à jour la valeur de l'état de probabilité de transition et définit l'état de cette ressource volontaire à déconnecter. Appelez-le (FTA Algorithme 6) en lui envoyant l'identifiant de cette ressource volontaire (Algorithme 1, ligne 35).

Le dernier cas concerne la déconnexion imprévisible, ce cas est détecté lorsque le switcher balancer reçoit un événement de dépassement de délai (time out) (ligne 38 de l'algorithme 1). Dans ce cas, le switcher balancer met à jour la valeur de l'état de probabilité de transition et définit l'état de cette ressource volontaire à déconnecter. Appelez-le (FTA Algorithme 6) en lui envoyant l'identifiant de cette ressource volontaire (Algorithme 1, ligne 40).

b) Description de l'algorithme 2 (Algorithme d'équilibrage de charge centralisé)

Lorsqu'un switcher balancer reçoit une demande RS_i d'un autre switcher balancer, l'algorithme 2 commence par sélectionner à la ligne 01 l'ensemble $S_{RS_j}^{VCL_i}$ (ensemble de

ressources volontaires contenant le service demandé). Afin de sélectionner la meilleure ressource en fonction des besoins de l'utilisateur (minimisant à la fois le temps de réponse et la probabilité d'échec) et de réguler la charge entre les ressources volontaires, une approche en deux étapes est conçue.

Dans la première étape l'algorithme 2 calcule la charge moyenne L_{VCL_i} de son sous-groupe volontaire VCL_i (algorithme 2 ligne 02) selon la formule (2) :

$$L_{VCL_i} = \frac{\sum_{j=1}^{[S_{RS_j}^{VCL_i}]} L_{VR_j}}{[S_{RS_j}^{VCL_i}]} \quad \ldots. (2)$$

$$L_{VR_j} = \sum_{i=1}^{[NRS_{VR_j}]} RS_{VR_j} \times T_i \quad \ldots. (3)$$

Pour calculer le temps de réponse nécessaire à l'exécution d'une requête RS_i en utilisant la ressource volontaire VR_j, nous proposons la formule (4) ci-dessous :

$$RT(RS_i)_{VR_j} = L_{VR_j} + T_{RS_i} \quad \ldots. (4)$$

Où T_{RS_i} est le temps d'estimation nécessaire pour exécuter le service de requête RS_i sur la ressource volontaire VR_j (selon CPU et RAM).

Après avoir obtenu la charge moyenne, il semble que certaines ressources volontaires soient surchargées et que les autres soient sous-chargées. À cette étape, l'algorithme 2 sélectionne parmi les ressources contenant le service demandé, celles qui sont sous-chargées (algorithme 2, ligne 03).

Dans la deuxième étape, l'algorithme de sélection (algorithme 3) est utilisé pour estimer les meilleures solutions (algorithme 2, ligne 04). Le pseudo-code de cet algorithme est bien détaillé dans la section suivante.

Les dernières étapes de cet algorithme sont expliquées comme suit :

- Envoyer la demande à la ressource volontaire sélectionnée VR_j (algorithme 2, ligne 04).
- Insérer la demande de l'utilisateur dans SRS_{VR_j} (ensemble de demandes en attente d'exécution sur la ressource VR_j appartenant au sous-groupe volontaire VCL_i) (algorithme 2, ligne 5).
- Mettre à jour la charge L_{VR_j} de la ressource volontaire sélectionnée VR_j (algorithme 2, ligne 06).

Algorithm 2: Centralized load balancing algorithm

Begin

01 $S_{RS_j}^{VCL_i} \leftarrow$ set of volunteer resources that contain the requested service

02 $L_{VCL_i} \leftarrow \frac{\sum_{j=1}^{[S_{RS_j}^{VCL_i}]} L_{VR_j}}{[S_{RS_j}^{VCL_i}]}$

03 Select from $S_{RS_j}^{VCL_i}$ the volunteer resources VR_j which their load L_{VR_j} is less than the average load of system L_{VCL_i}

04 Call The (SA) algorithm algorithm 3 to find the best solution and Send the request to the selected one

05 Add the request to SRS_{VR_j} (set of requests awaiting execution at resource VR_j belonging to the volunteer sub-groups VCL_i

06 Update (L_{VR_i})

End algorithm

c) Description de l'algorithme 3 (Algorithme de sélection)

Dans cet algorithme, nous proposons un mécanisme permettant de réduire le nombre des ressources volontaires à chaque appel de récursivité afin de trouver la solution optimale.

L'algorithme de sélection est la phase la plus importante de notre travail. Avant de commencer à trouver la meilleure solution, nous utilisons l'algorithme quicksort pour ordonner les deux tableaux : le premier concerne la probabilité de défaillance et le second tableau est utilisé pour stocker le temps de réponse nécessaire de chaque ressource volontaire (lignes 27 et 28 de la SA). Après cela, nous procédons à une normalisation de l'échelle entre les deux tableaux, puis nous appelons la fonction de récursivité (ligne 33 de l'algorithme SA) pour obtenir la meilleure solution, qui prend en compte les paramètres suivants :

1. ***Set_population***: est un ensemble de ressources volontaires obtenues par l'intersection des membres appartenant aux deux tableaux. Le premier est le tableau de temps de réponse où le temps de réponse des membres est inférieur ou égal à *DistanceVR* et le second est le tableau de probabilité d'échec où la défaillance des membres est inférieure ou égale à *DistanceVR*.

Notons que le temps de réponse calculé pour chaque ressource volontaire est stocké dans le tableau Temps de réponse et calculé selon la formule (4).

2. ***DistanceVR***: est une valeur calculée, comme le montre la Figure 3.4, la distance entre le point de la ressource volontaire ($RTVR_{IDSelectVR}$, $FVR_{IDSelectVR}$) et le point d'origine (0,0) est calculée selon la formule suivante : (5)

$$Distance_{IDSelectVR} \leftarrow \sqrt{(RTVR_{IDSelectVR})^2 + (FVR_{IDSelectVR})^2}; \ldots. (5)$$

3. ***Min***: comme le montre la Figure 3.5, le Min représente la différence entre l'angle de l'exigence d'utilisateur ϒ et l'angle de l'élément choisi parmi les éléments de même distance α, β.
4. ***Random_array***: il contient le membre choisi de manière aléatoire. Ce tableau sera vide lorsque le membre choisi aura la plus petite distance.
5. ***Index_VR***: est l'ID de la meilleure ressource volontaire trouvée.

Comme le montre la Figure 3.4, la fonction de récursivité sélectionne au hasard, à tout appel, un membre de la population (ligne SA 05) afin de calculer sa distance et sa différence d'angle (ligne SA 06,07) pour enfin les comparer à la solution précédente. Comme montre la Figure 3.5, si la distance calculée est égale à la distance précédente (ligne SA 09), la fonction de récursivité doit comparer l'angle de différence calculé (ligne SA 10) afin de choisir l'angle de différence minimal pour répondre aux besoins de l'utilisateur.

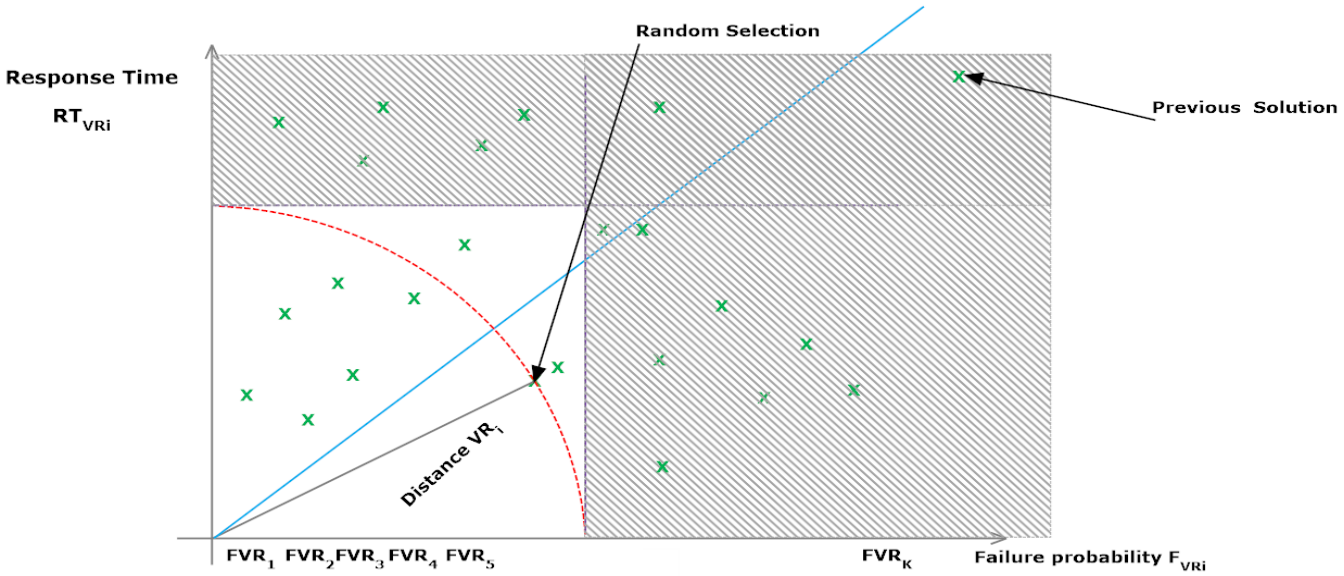

Figure 3.4 Réduire le nombre de la population

Si le membre sélectionné a l'angle de différence minimal, l'algorithme proposé met à jour la valeur de Min (ligne 11 de la SA), puis le membre choisi devient la meilleure solution (ligne 12 de la SA). Sinon, si elle est extrêmement inférieure à la distance calculée précédemment (ligne SA 14), alors le membre choisi devient le meilleur (ligne SA 17).

Dans ce cas, la fonction de récursivité met à jour à la fois la distance et l'angle (lignes SA 15, 16). Elle crée aussi un nouveau tableau aléatoire (ligne SA 19) et met à jour l'ensemble

de la population comme illustré à la Figure 3.4 obtenue par l'intersection entre les membres appartenant aux deux tableaux : le premier est le tableau de temps de réponse où leur temps de réponse devrait être inférieur à la nouvelle *DistanceVR* calculée. Le second est le tableau des probabilités de défaillance, dans lequel leurs probabilités de défaillance devraient être inférieures à la nouvelle *DistanceVR* calculé (ligne SA 20). Après cela, la fonction de récursivité ajoute le membre choisi de manière aléatoire au tableau aléatoire (ligne SA 22) et appelle à nouveau la fonction de récursivité avec les nouveaux paramètres (ligne SA 23).

Cette fonction s'arrêtera à la satisfaction de l'une des conditions suivantes:

- S'il n'y a qu'un seul membre dans l'ensemble de la population.
- Si le tableau de population défini et le tableau aléatoire ont le même membre.

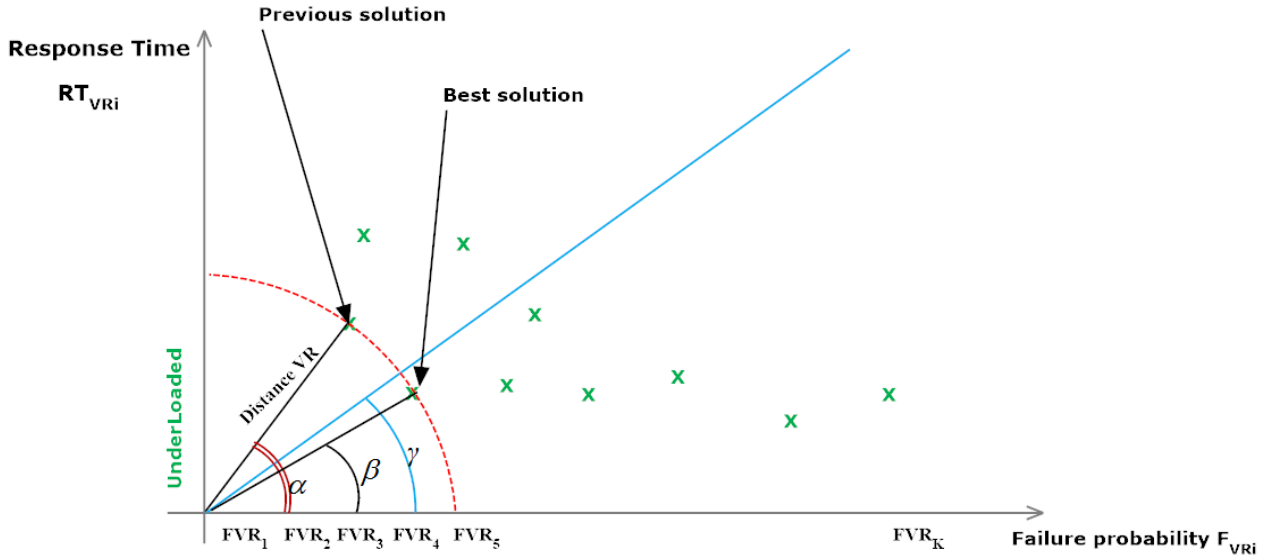

Figure 3.5 Choisissez la meilleure solution lorsque les membres ont la même distance

Algorithm 3 Selecting Algorithm SA

Algorithm SA
Function
01 *Find_Best_Solution* (*Set_population*[], *DistanceVR*: *float*, *Min*: *float*, *Random_array*[], *Index_VR*) **:entier;**
02 **if**$((length(Set_{Population}) == 1)$ ***or*** $(length(Set\_Population) == length(Random))$ **then**
03 $Find\_Best\_Solution \leftarrow Index\_VR$;
04 **else**
05 $IDSelectVR \leftarrow Set\_population[RandomID]$;
06 $Distance_{IDSelectVR} \leftarrow \sqrt{(RTVR_{IDSelectVR})^2 + (FVR_{IDSelectVR})^2}$;
07 $Sin_{IDSelectVR} \leftarrow \frac{(TRVR_{IDSelectVR})}{Distance_{IDSelectVR}}$;
08 **if** $(Distance_{IDSelectVR} \leq DistanceVR)$ **then**
09 **if** $(Distance_{IDSelectVR} = DistanceVR)$ **then**
10 **if** $(|(Sin\ weightRequestFailure \times 90\ - Sin_{IDSelectVR})| < Min)$ **then**
11 $Min = |(Sin\ weightRequestFailure \times 90\ - Sin_{IDSelectVR})|$;
12 $Index\_VR \leftarrow\ IDSelectVR$;
13 **end if**
14 **else**
15 $DistanceVR \leftarrow Distance_{IDSelectVR}$;
16 $Min = |(Sin\ weightRequestFailure \times 90\ - Sin_{IDSelectVR})|$;
17 $Index\_VR \leftarrow\ IDSelectV$;
18 **end if**
19 $Random\_array = new\ Random\_array[]$;
20 $Set\_population \leftarrow RTVR_{Distance_{IDSelectVR}} \cap FVR_{IDistance_{IDSelectVR}}$;
21 **end if**
22 $Random\_array.ADD(RandomID)$;
23 $Find\_Best\_Solution \leftarrow Find\_Best\_Solution\ (Set\_population, DistanceVR, Min, Random\_array[], Index\_VR)$;
24 **end if**
end
Begin
25 $BestVR \leftarrow -1$;
26 $Min \leftarrow Min = Sin\ weightRequestFailure \times 90$;
27 QuickSort (*FVR*);
28 QuickSort (*RTVR*);
29 Scale (*RTVR*);
30 $DistanceVR \leftarrow \sqrt{(RTVR_n)^2 + (FVR_n)^2}$;
31 $Set\_population \leftarrow RTVR_i \cap FVR_i$;
32 $Random\_array = new\ Random\_array[]$;
33 $BestVR \leftarrow\ Find\_Best\_Solution\ (Set\_population[], DistanceVR, Min, Random[], BestVR)$;
End algorithm

d) Description de l'algorithme 4 (Algorithme d'équilibrage de charge décentralisé)

Pour équilibrer la charge de manière décentralisée, l'algorithme 4 commence par calculer la charge moyenne du système (L_{System}), (cette étape sera bien détaillée dans section suivante).

Ensuite, l'algorithme sélectionne les sous-groupes volontaires actifs sous-chargés, comme indiqué dans la Figure 3.6, ainsi que les classes dans un tableau de la collection RS ($Load$ [ ]) dans un ordre croissant (étapes de l'algorithme 4, lignes 02 à 08).

Les sous-groupes volontaires actifs signifient que la moyenne de la probabilité d'échec de leurs ressources volontaires est supérieure à un pourcentage défini dans le système. Cet état d'activité est calculé comme suit :

$$Active_{VCL_i} = \frac{\sum_{j=1}^{n}(F_{VR_j})^{-1}}{n} \quad \ldots. \ (6)$$

Où n est le nombre de ressources volontaires appartenant au sous-groupe volontaire VCL_i.

F_{VR_j} est la probabilité d'échec de la ressource volontaire VR_j.

L'étape suivante va de la ligne 09 à la ligne 16 de l'algorithme 4. Dans cette étape, le switcher balancer sélectionne dans le tableau Load [] le premier switcher balancer sous-chargé et actif.

Si le switcher balancer est différent de l'actuel (Algorithme 4, ligne 09), la demande de l'utilisateur est dans ce cas acheminé vers le switcher balancer sélectionné (Algorithme 4, ligne 10). Dans le second cas, l'actuel switcher balancer appelle l'algorithme 2 (ligne 12 de l'algorithme 4), puis met à jour sa charge (lignes 13 et 14 de l'algorithme 4) et collabore avec les autres switchers balancers appartenant à son quorum afin de partager sa nouvelle charge (Algorithme 4, ligne 15).

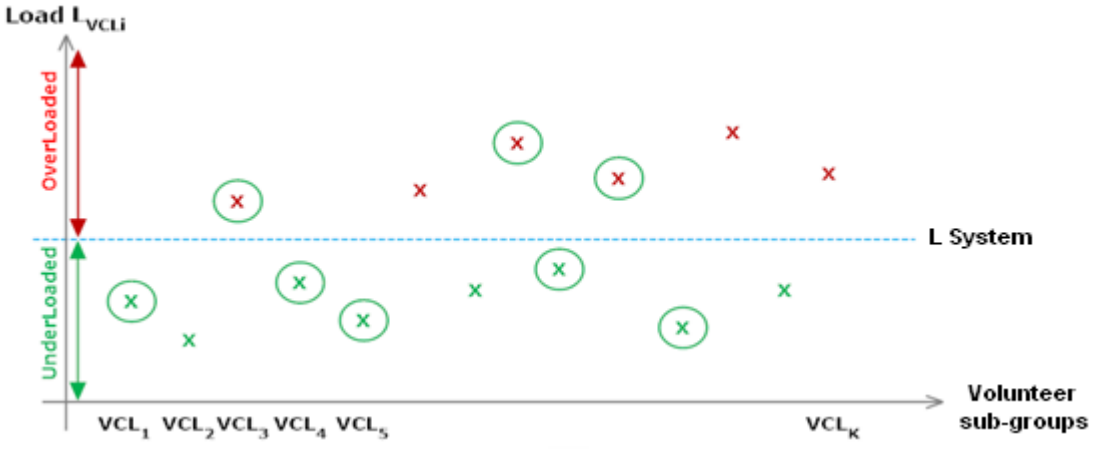

Figure 3.6 État du système

Algorithm 4: Decentralized load balancing algorithm

Begin
01 Algorithm 5; // Call the Algorithm 5 in order Calculate the average load of the system
02 **For** i ← 1 to K **do**
03 **if** ((L_{VCL_i}<L_{SUP_VCL}) and (VCL_i *is Active*)) **then**
04 Load[j] ← L_{VCL_i};
05 j++;
06 **end if**
07 **end for**
08 Arrange *Load*[*j*] in increasing order;
09 **if** *Minvcl* ≠ *This switcher balancer* **then**
10 *Send* (*Rq*, *Minvcl*);
11 **else**
12 *Algorithm* 2(*Rq*);
13 *Update* (L_{VCL_i}) of this switcher balancer;
14 $L[i] \leftarrow L_{VCL_i}$;
15 *Broadcast*(L_{VCL_i}) to the quorum that the switcher balancer belong to it;
16 **end if**
End algorithm

e) Description de l'algorithme 5 (Calcul de la charge moyenne du système en fonction du système de quorum)

Pour équilibrer la charge de manière décentralisée entre les switchers balancers de la couche des switchers, nous devons calculer la charge moyenne du système.

Pour tout besoin de calculer la charge moyenne du système, nous devons partager les informations de charge de tous les switchers balancers entre eux. Nous avons utilisé un système de quorum afin de réduire le nombre de messages échangés entre les switchers balancers. Tout switcher balancer doit calculer la charge moyenne, envoyer un message de quorum uniquement aux switchers balancers appartenant à son quorum et possédant la propriété d'intersection. Dans le meilleur des cas, la complexité de cet algorithme est égale au nombre de switchers balancers d'intersection. Par contre, dans le pire des cas, cette complexité est égale au nombre de switcher balancers contenus dans le quorum le plus volumineux (concernant le nombre de switchers balancers).

$O(L)$ Correspond à la complexité de l'algorithme, où $1 \leq L \leq [Q]$, $[Q]$ est le nombre de switchers balancers dans le quorum Q.

Comme indiqué dans le diagramme de séquence ci-dessous (voir la Figure 3.7):

Après avoir reçu une demande de l'utilisateur, le switcher balancer concerné envoie un message de quorum aux switchers balancers qui appartiennent à son quorum et qui ont la propriété d'intersection, afin d'obtenir une réponse sous forme des listes contenant les switcher balancers avec leur charge. Ensuite, le switcher balancer concerné regroupe toutes les réponses reçues avec l'opérateur d'union. Comme on l'a déjà noté, l'opérateur d'union élimine la répétition. Nous obtenons ensuite une liste de tous les switchers balancers du système avec leur charge. De plus, nous calculons la charge moyenne du système selon la formule suivante (7) :

$$L_{System} = \frac{\sum(RS)}{[RS]} \quad \ldots. (7)$$

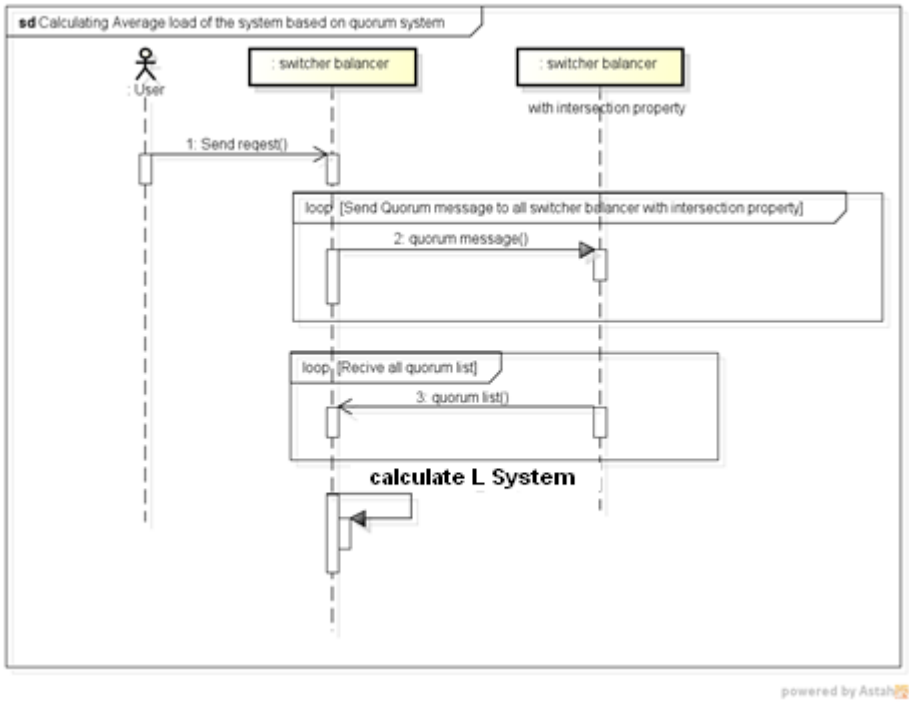

Figure 3.7 le diagramme de séquence (comment calculer la charge moyenne du système en fonction du système de quorum)

L'exemple présenté dans la Figure 3.8 explique plus en détail cette opération. Lorsque le switcher balancer 1 doit calculer la charge moyenne du système, il envoie un message de quorum uniquement au switcher balancer appartenant à son quorum et possédant les propriétés d'intersection. Dans cet exemple, il envoie un message aux switchers balancers **4** et **5**. Après cela, il reçoit une réponse sous la forme d'un ensemble des switchers balancers avec la charge de chacun.

- De la part du switcher balancer 4, il reçoit deux listes :

N° Switcher Balancer	La Charge (ms)
4	380
6	350
7	320
8	370

N° Switcher Balancer	La Charge (ms)
4	380
2	360
1	330
3	320
5	340

- Et de la part du switcher balancer 5, il reçoit deux listes :

N° Switcher Balancer	La Charge (ms)
5	340
12	310
11	380
10	360
9	350
8	370

N° Switcher Balancer	La Charge (ms)
5	340
3	320
1	330
2	360
4	380

Après cela, on réunit toutes ces listes et on élimine la répétition, le résultat est montré dans le tableau ci-dessous Tableau 3.1, maintenant nous pouvons calculer la charge moyenne du système selon la formule (7) : $L_{System} = 347.5$

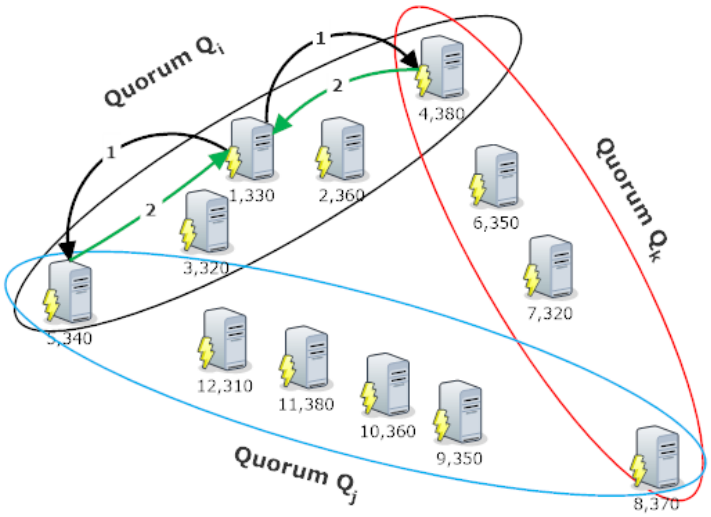

Figure 3.8 Exemple de système de quorum montrant les switchers balancers et leur charge

N° Switcher Balancer	**La Charge(ms)**
1	330
2	360
3	320
4	380
5	340
6	350
7	320
8	370
9	350
10	360
11	380
12	310

Tableau 3.1 Liste contenue la charge de tous les switchers balancers du système

Comme indiqué dans l'exemple précédent (Figure 3.8), nous utilisons seulement deux messages **02** pour calculer la charge du système pour **12** switchers balancers.

f) Description de l'algorithme 6 (Algorithme de tolérance aux pannes)

Lorsqu'un switcher balancer détecte la défaillance de la ressource volontaire (ligne 33 ou ligne 38 de l'algorithme 1), il appelle l'algorithme 6 en transmettant l'identificateur de cette ressource volontaire entant que paramètre (ligne 35 et ligne 40 de l'algorithme 1). Lorsque l'algorithme 6 est appelé, il supprime l'identifiant de cette ressource de la liste des ressources volontaires disponibles (algorithme 6, ligne 01), puis met à jour la charge de son sous-groupe volontaire (algorithme 6, ligne 02) ainsi que de son tableau de charge (algorithme 6, ligne 03). Enfin, l'algorithme redistribue toutes les demandes en attente d'exécution sur cette ressource volontaire et les traite en appelant l'algorithme d'équilibrage de charge décentralisé Algorithme 1 (algorithme 6, lignes de 04 à 07).

Algorithm 6: FTA : Fault Tolerance Algorithm

Begin FTA (VR_i)
01 Remove VR_i from the available volunteer resources list SVR_i;
02 $L_{VCL_i} \leftarrow L_{VCL_i} - L_{VR_i}$;
03 Load[j] $\leftarrow L_{VCL_i}$;
04 **while** RS_{VR_j} *is not empty* **do**
05 retrieve Rq request fromRS_{VR_j};
06 Algorithm1(Rq);
07 **end while**
end algorithm

3.4.2 Modèle de défaillance

Lors de l'exécution du système, les ressources volontaires peuvent rejoindre ou quitter le système à tout moment. Dans l'approche que nous avons définie, il existe deux types de composants : les ressources volontaires et la communication en réseau. Chacun de ces composants peut échouer lorsque le système fonctionne (Ledmi et al., 2018). Dans notre travail, nous nous concentrons sur l'échec des ressources volontaires. Nous supposons qu'une ressource volontaire fonctionne toujours correctement ou ne fonctionne pas du tout (elle est en panne, *halt-crash*). Afin de tolérer ce type d'échec, le switcher balancer doit redistribuer les tâches en attente d'exécution sur cette ressource volontaire entant que nouvelles demandes à l'aide de l'algorithme 1.

Nous distinguons deux situations de déconnexion: la déconnexion prévisible et imprévisible:

1. Déconnexion prévisible

Lorsqu'une ressource volontaire choisit délibérément de quitter le sous-groupe volontaire, elle envoie un message de demande de déconnexion à son switcher balancer.

2. Déconnexion imprévisible

La déconnexion imprévisible peut survenir lors de l'exécution de la demande, lors de l'acheminement de la demande, ainsi que pendant les phases de réponse.

Dans notre travail, nous traitons les deux situations décrites ci-dessous.

a) Détection d'échec

Habituellement, les échecs sont détectés périodiquement par des messages de pulsation (heartbeat messages) (Aguilera, Chen, & Toueg, 1999) ou à la demande par des messages de ping-pong (Antoniu, Deverge, & Monnet, 2006). Nous prenons en compte les deux techniques.

b) Algorithme PING

Le but de cet algorithme est d'envoyer périodiquement un message Ping afin de détecter la défaillance de la ressource volontaire. Chaque switcher balancer envoie un message ping à toutes les ressources volontaires connectées qui lui appartiennent, et initialise un délai d'attente de chaque ressource volontaire (PING, lignes 13 à 16). Après cela, il calcule les états de probabilités de transitions de toutes les ressources volontaires. Enfin, il évalue la

probabilité de défaillance (PING, ligne 27) de chaque ressource volontaire selon la formule (09) et calcule la valeur active (PING, ligne 28) selon la formule (06) (le pseudo-code de cet algorithme est donné ci-dessous).

Ping : Sending Ping Periodically in order to calculate volunteer resource failure probability

Begin

01 p_i is the probability that the volunteer resource VR_i when it is still connected;
02 r_i is the probability that the volunteer resource VR_i when it is deliberately decide to leave the system;
03 t_i is the probability that the volunteer resource VR_i when it is fails;
04 q_i is the probability that the volunteer resource VR_i when it is still disconnected;
05 $1 - q_i$ is the probability of the volunteer resource VR_i when it is connected again;
06 C is the number of cycles;
07 T is the time to send another ping;
08 $P_{ij}[VR_k]$ is an incremental variable used to count the number of change from the state i to state j, where $i \in \{C, F\}$, and $j \in \{C, D, F\}$ the time to send another ping;
09 **While** true
10 Create Msg message type of ping;
11 **Repeat**
12 Wait(T);
13 **For** $i \leftarrow 1$ to M do
14 **If** (getState(VR_i)==Connected) **then**
15 Send(Msg,VR_i);
16 InitializedTime(VR_i);
17 **Else**
18 $P_{FF}\ VR_i \leftarrow P_{FF}\ VR_i + 1$;
19 **End if**
20 **End for**
21 $C \leftarrow C - 1$;
22 **Until** $(C = 0)$;
23 $p_i \leftarrow P_{CC}\ VR_i \,/\, (P_{CC}\ VR_i + P_{CD}\ VR_i + P_{CF}\ VR_i)$;
24 $r_i \leftarrow P_{CD}\ VR_i \,/\, (P_{CC}\ VR_i + P_{CD}\ VR_i + P_{CF}\ VR_i)$;
25 $t_i \leftarrow P_{CF}\ VR_i \,/\, (P_{CC}\ VR_i + P_{CD}\ VR_i + P_{CF}\ VR_i)$;
26 $q_i \leftarrow P_{FF}\ VR_i \,/\, (P_{FF}\ VR_i + P_{FC}\ VR_i)$;
27 Calculating γ_i for every Volunteer resource VR_i, according to equation (09);
28 Calculating the $Active_{VCL}$ according to equation (6);
28 **End while**
End algorithm

c) Le modèle de chaîne de Markov discret

Il existe plusieurs modèles de chaîne de Markov dans la littérature. Dans notre thèse, nous nous concentrons sur une chaîne de Markov à temps discret (DTMC[19]) (Tijms, 2003) . Conformément à notre proposition, nous supposons que S est un ensemble d'états concernant l'espace de notre modèle ;

S = {C, D, F}. Nous définissons ces états de l'espace comme suit :

C_i: est l'état de la ressource volontaire VR_i lorsqu'elle est connectée au système.

D_i: est l'état de la ressource volontaire VR_i lorsqu'elle décide délibérément de quitter le système.

F_i: est l'état de la ressource volontaire VR_i lorsqu'elle est déconnectée (en échec).

De plus, chaque ressource volontaire est caractérisée son propre modèle de chaîne de Markov. Nous considérons les probabilités de changer d'état à un autre état comme suit :

p_i: est la probabilité que la ressource volontaire VR_i soit encore connectée.

r_i: est la probabilité de la ressource volontaire VR_i lorsqu'elle décide délibérément de quitter le système.

t_i: est la probabilité de la ressource volontaire VR_i en cas d'échec.

q_i: est la probabilité que la ressource volontaire VR_i soit encore déconnectée.

$1 - q_i$: est la probabilité que la ressource volontaire VR_i rejoigne le système.

Un diagramme d'état pour leur modèle est présenté à la Figure 3.9, utilisant un graphe orienté pour représenter les transitions d'état.

[19] DTMC: Discrete Time Markov Chain

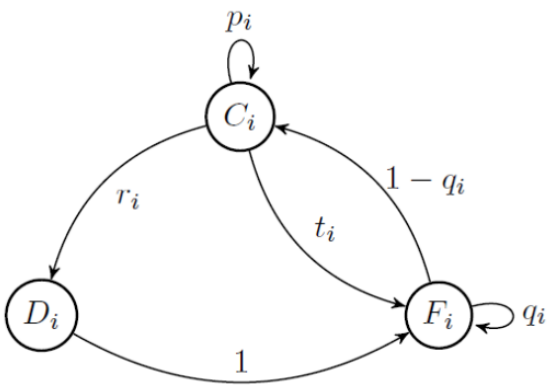

Figure 3.9 Modèle de Markov.

Notre modèle de chaîne de Markov est représenté par la matrice de probabilités de transitions (TPM). Cette dernière a 3 × 3 éléments. Ici, les lignes représentent l'état d'origine de la transition et les colonnes représentent les états dans lesquels la transition est effectuée. Chaque cellule d'un module TPM représente la probabilité de passer d'un état à l'autre, comme indiqué ci-dessus. Comme dans tout TPM stochastique, les valeurs de transition de toutes les colonnes d'une ligne doivent totaliser à 1,0. La matrice de transition de la ressource volontaire (VR_i).

$$TPM(VR_i) = \begin{bmatrix} p_i & r_i & t_i \\ 0 & 0 & 1 \\ 1-q_i & 0 & q_i \end{bmatrix}$$

Dans ce modèle, tous les états appartiennent à la même classe récurrente, selon les probabilités d'état stable (Steady-State Probabilities) (Tijms, 2003), la probabilité de passer de l'état i à l'état j en n étapes est la suivante :

$r_{ij}(n) = \sum_k r_{ik}\,(n-1)p_{kj}$ nous prenons la limite comme $n \to \infty$ nous avons: Les équations d'équilibre :

$$\begin{cases} \pi_j = \sum_k \pi_k\, p_{kj} \\ \sum_j \pi_j = 1 \end{cases} \quad \text{.... (8)}$$

Pour calculer λ_i la probabilité d'échec de la ressource volontaire VR_i, $\lambda_i = \pi_{F_i}$ selon l'équation (8), on a :

$$\begin{cases} \pi_{F_i} = \pi_{C_i} t_i + \pi_{F_i} q_i + \pi_{D_i} \\ \pi_{C_i} = \pi_{C_i} p_i + \pi_{F_i}(1-q_i) \\ \pi_{D_i} = \pi_{C_i} r_i \\ \pi_{F_i} + \pi_{C_i} + \pi_{D_i} = 1 \end{cases}$$

Nous obtenons ensuite la probabilité d'échec de la ressource volontaire à l'heure mentionnée ci-dessus :

$$\lambda_i = \pi_{F_i} = \frac{p_i - 1}{p_i - 2 + q_i - r_i + (r_i \times q_i)} \quad \ldots. \ (9)$$

3.5 Conclusion

Dans ce chapitre, nous avons présenté notre contribution pour la tolérance aux pannes dans les systèmes de calculs volontaires. Nous avons proposé une architecture permettant d'équilibrer la charge entre les ressources en tenant en compte les contraintes imposées par les besoins des utilisateurs en systèmes de calculs volontaires.

Dans cette architecture, les demandes sont directement envoyées au switchers balancers, ce qui nécessite l'optimisation des deux critères : le temps de réponse et la probabilité d'échec. Ces deux critères sont contradictoires et ont un impact négatif sur l'équilibrage de charge dans le système. Pour régler ce problème, nous avons proposé quatre algorithmes. Le premier algorithme proposé est utilisé pour équilibrer la charge globale du système entre les sous-groupes volontaires de manière distribuée. Le deuxième sert à équilibrer la charge entre les ressources volontaires de chaque sous-groupe volontaire de manière centralisée. Le troisième est utilisé pour satisfaire les besoins des usagers. De plus, le dernier algorithme est utilisé pour estimer la probabilité d'échec des ressources volontaires en utilisant un modèle de chaîne de Markov.

Dans le chapitre suivant, pour valider notre architecture et nos algorithmes proposés, nous avons effectué quelques expérimentations en utilisant le simulateur PeerSim. Le but d'utiliser un simulateur au lieu d'une véritable plate-forme est que nous pouvons gaspiller beaucoup de puissance de calcul et de temps. Pour cela, un banc de test basé sur la simulation éviterait ces problèmes.

Chapitre 4 SIMULATIONS ET DISCUSSIONS DES RESULTATS

4.1 Introduction

Dans ce chapitre, nous décrirons les résultats et évaluerons la performance de notre approche à travers la simulation utilisant Peersim (Montresor & Jelasity, 2009). Nous allons d'abord décrire l'environnement de simulation. Ensuite nous présenterons et analyserons les résultats de nos scénarios et expérimentations. Enfin, à partir de ces analyses, nous prouverons que notre approche est efficace et avantageuse en comparaison avec des travaux récents.

4.2 Environnement de simulation

Dans cette section, nous décrivons les outils utilisés pour simuler notre approche. La simulation est une représentation simplifiée du système, facile à réaliser et requiert moins de ressources que l'implémentation, ce qui favorise l'évaluation d'un système à grande échelle.

a) Langage de développement

La version de PeerSim 1.05 que nous utilisons est implémentée en langage JAVA, le choix du langage de développement s'impose donc logiquement à nous. Le langage JAVA présente de nombreux avantages. En effet, il est multi-plate-forme ce qui s'accorde judicieusement avec un déploiement sur un réseau Peer-2-Peer.

b) Environnement de développement

Eclipse J2EE (Java 2 Entreprise Edition) est un IDE (Integrated Developpement Environement), une plate-forme de développement entièrement en java. Il offre de nombreuses facilités : une interface de lancement d'application. Eclipse est un environnement ouvert et extensible. L'interface offerte permet une customisation complète pour s'adapter au développeur. Par ailleurs, les plug-ins existants permettent d'intégrer tout type d'applications annexes.

c) Le simulateur PeerSim

Peersim est un logiciel libre sous la licence open source GPL, développé en langage Java, destiné à simuler le mode de fonctionnement des réseaux P2P. Il permet alors de concevoir

et de tester toute sorte d'algorithmes de réseaux P2P dans un environnement dynamique. Le code source peut être téléchargé à partir de site web ("PeerSim,"). Ce logiciel est caractérisé essentiellement par :

- Un passage à l'échelle (plus d'un million de nœuds).
- Une très grande dynamicité (arrivée/départ) des nœuds.
- Une architecture ouverte et composée de modules.

PeerSim ignore les détails de l'implémentation de la pile de protocoles TCP/IP et ne modélise pas les messages échangés entre protocoles, ceci dans l'optique de diminuer la charge CPU et la grande quantité de mémoires que la communication entre protocoles requiert. Néanmoins le développeur peut penser à les implémenter. Pour plus de détail, voir l'annexe A.

Essentiellement, PeerSim supporte deux types de simulation qui sont :

- ***La simulation par cycle*** : elle s'effectue dans un ordre séquentiel. Dans chaque cycle, chaque nœud peut exécuter son comportement (protocole).
- ***La simulation par évènement*** : elle supporte la concurrence ; un ensemble d'évènements est planifié et les protocoles d'un nœud sont exécutés en fonction des évènements survenus.

Dans notre travail, on s'intéresse aux deux types de simulation. La simulation par cycle pour gérer la dynamicité (arrivée/départ) des nœuds, et la simulation par évènement qui nous a aidés à mettre en œuvre les algorithmes pour chaque évènement. Nous allons laisser évoluer le réseau jusqu'à ce que tous les événements soient terminés. Il nous suffira d'analyser l'état du réseau à la fin de la simulation pour en tirer les bonnes conclusions.

En PeerSim, tout est paramétrable, pour simuler et observer tout type de réseau P2P, les développeurs de PeerSim mettent ainsi à notre disposition trois types de classes :

- ***Protocoles*** : ils définissent un comportement de nœud.
- ***Dynamics*** : ils permettent de définir le réseau.
- ***Observers*** : ils permettent d'observer la simulation.

Toutes les données utiles au simulateur sont regroupées dans un fichier de configuration (Figure 4.1) où l'on définit les variables du réseau, déclare la structure d'un nœud (les

protocoles qui vont s'y exécuter à chaque évènement), les protocoles, les Dynamics et les Observers que l'on veut utiliser.

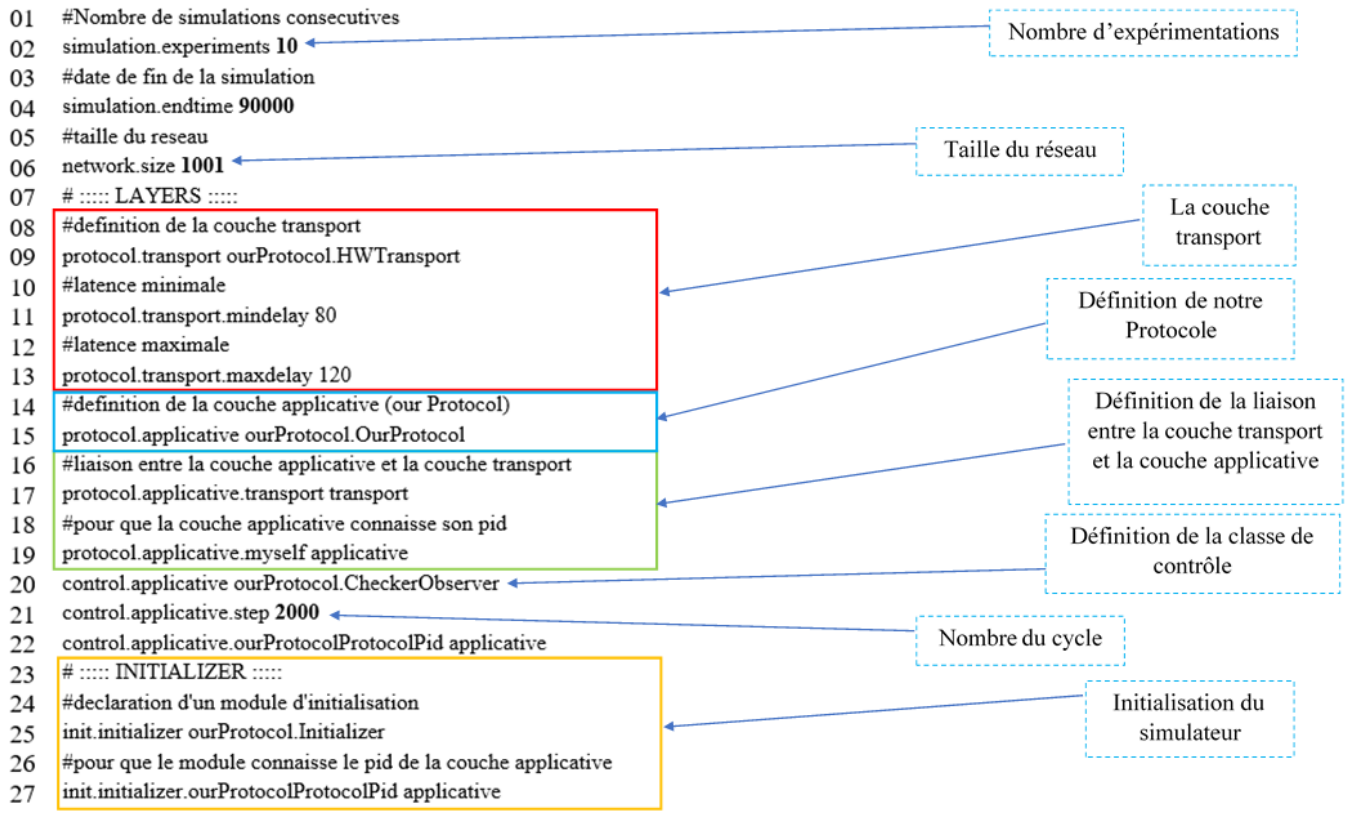

Figure 4.1 Fichier de configuration de la simulation.

1. Les protocoles

Comme vu plus haut, un protocole est un comportement de nœuds. En effet, dans un réseau P2P que l'on veut simuler avec PeerSim, chaque nœud dispose d'une pile de protocoles qui vont être exécutés à chaque évènement de simulation, et ceci dans leur ordre de déclaration dans le fichier de configuration. Bien évidemment, tout est mis à notre disposition pour créer nos propres protocoles. Chaque protocole est décrit dans la classe Java qui porte son nom.

Lors du lancement du simulateur, PeerSim crée un squelette de nœud avec une instance de tous les protocoles déclarés dans le fichier de configuration, puis clone ce squelette autant de fois que le nombre de nœuds dans le réseau.

2. Les Dynamics

Le but est d'introduire un dynamisme dans le réseau, essentiellement sous deux formes : d'une part, initialiser le réseau, c'est-à-dire initialiser les protocoles présents sur les nœuds

du réseau et définir les quorums sur le réseau, et d'autre part gérer et simuler arriver/départ des nœuds.

Tout comme pour les protocoles, nous pouvons créer nos propres Dynamics. Il suffit de créer une classe Java implémentant l'interface Dynamics fournie dans PeerSim. Le fichier de configuration peut contenir deux types de class de Dynamics :

- ✓ Le type Initializer sera exécuté une seule fois et au début de la simulation.
- ✓ Le type standard sera exécuté au début de chaque cycle de simulation class.

Pour effectuer la simulation nous avons implémenté plusieurs classes, comme on peut le voir dans la Figure 4.2.

- ❖ Initializer: cette class est une implémentation de la class Control, correspondant à l'initialisation du réseau, et à la construction de quorums. La class sera exécuté une seule fois et au début de la simulation.
- ❖ OurProtocol : cette class est une implémentation du class EDProtocol qui représente la simulation par évènement.

❖ Matrix : cette class implémente la matrice des chaines des Markov.

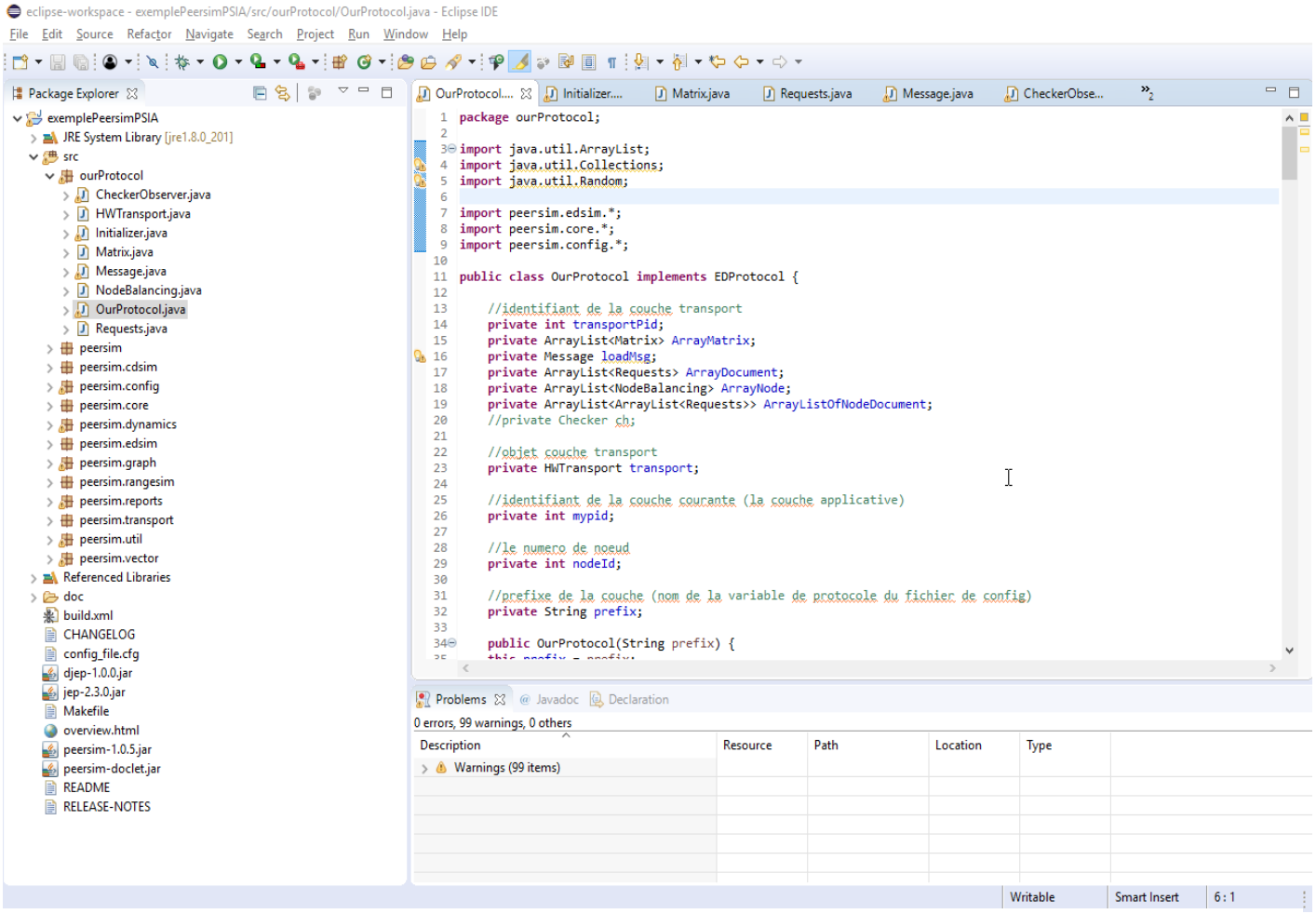

Figure 4.2 Les classes implémentées.

Avec la console (Figure 4.3) d'Eclipse, on peut voir tous les traitements des algorithmes dans chaque nœud (ressource volontaire ou switcher balancer). Tous ces traitements sont enregistrés dans un fichier "Log.txt" dans le dossier de sauvegarde.

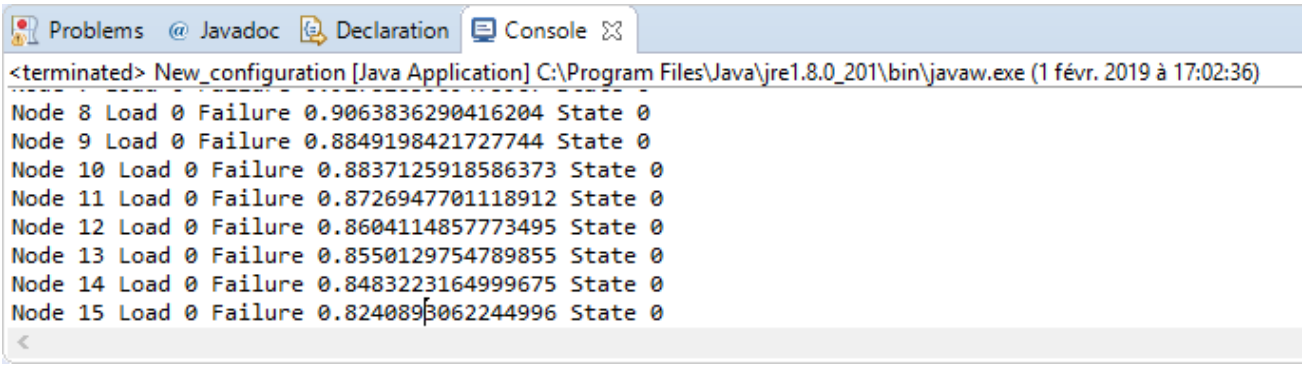

Figure 4.3 Exemple de traitement des algorithmes au cours de la simulation.

4.3 Résultats de la simulation

Les expériences sont exécutées sur une machine Intel Core i5, 2,5 GHz et Windows 10 équipé de 8 Go de mémoire principale. Chaque résultat est la moyenne d'environ 10 expériences. L'environnement utilisé est Eclipse 2018-12.

4.3.1 Scénario 1 : l'impact d'augmenter le nombre de demandes d'utilisateurs sur le temps de réponse

Dans ce scénario (voir Figure 4.4), nous définissons la configuration du système comme suit :

- Trois ensembles de quorums, chaque quorum contient 17 switchers balancers sur un total de 48 switchers balancers.
- Nous fixons le nombre de ressources volontaires à 100 pour chaque switcher balancer.
- Pour obtenir l'impact des demandes des utilisateurs sur le temps de réponse, nous faisons varier le nombre de demandes d'utilisateur de 100 à 20 000.

Nous avons constaté que le temps de réponse passe de 115 à 570 lorsque le nombre de demandes augmente de 100 à 500 et continu d'augmenter jusqu'en 1110, lorsque les demandes appliquées au système atteignent 1 000 demandes. Ce temps de réponse sera stable et moyen à 1100 lorsque le nombre de requêtes varie de 1000 à 20 000. Dans ce scénario, nous concluons que l'algorithme proposé garantit l'évolutivité, car lorsque le nombre de requêtes est supérieur à 1000, le temps de réponse moyen sera stable et autour de 1100.

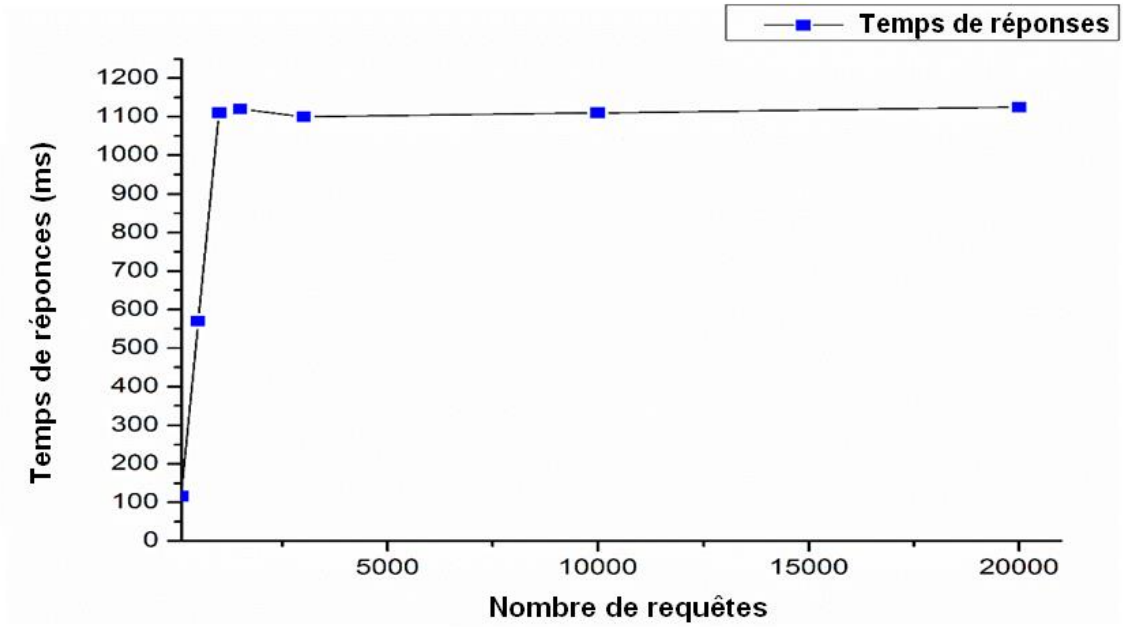

Figure 4.4 Scénario 1 : Variation du nombre de requêtes

4.3.2 **Scénario 2 : Efficacité des algorithmes d'équilibrage de charge**

Dans ce scénario, nous devons tester l'efficacité des deux algorithmes d'équilibrage de charge de deux manières (décentralisée et centralisée) :

a) **Le premier scénario**

Concerne l'algorithme de manière décentralisée (Algorithmes 4), les paramètres de ce scénario sont détaillés comme suit :

- Nous fixons les ensembles de quorums à trois, le nombre de switcher balancer à chaque quorum à 20 (le total du système est de 57 switchers balancers).
- Les ressources volontaires de chaque switcher balancer sont réglées sur 100 et les demandes des utilisateurs sur 3000.

Dans ce cas d'expérience, nous initialisons d'abord, de manière arbitraire, tous les switchers balancers avec une charge de manière aléatoire, comme illustrés à la Figure 4.5 (barre noire). Ensuite, nous lançons la simulation afin de montrer l'efficacité de l'algorithme 4. À la fin de la simulation, nous concluons que l'algorithme 4 proposé est efficace, car la différence de charge entre les switchers balancers insuffisamment chargés et surchargés n'est que de 30 % (barre rouge), alors que cette différence est de 380 à la phase d'initialisation.

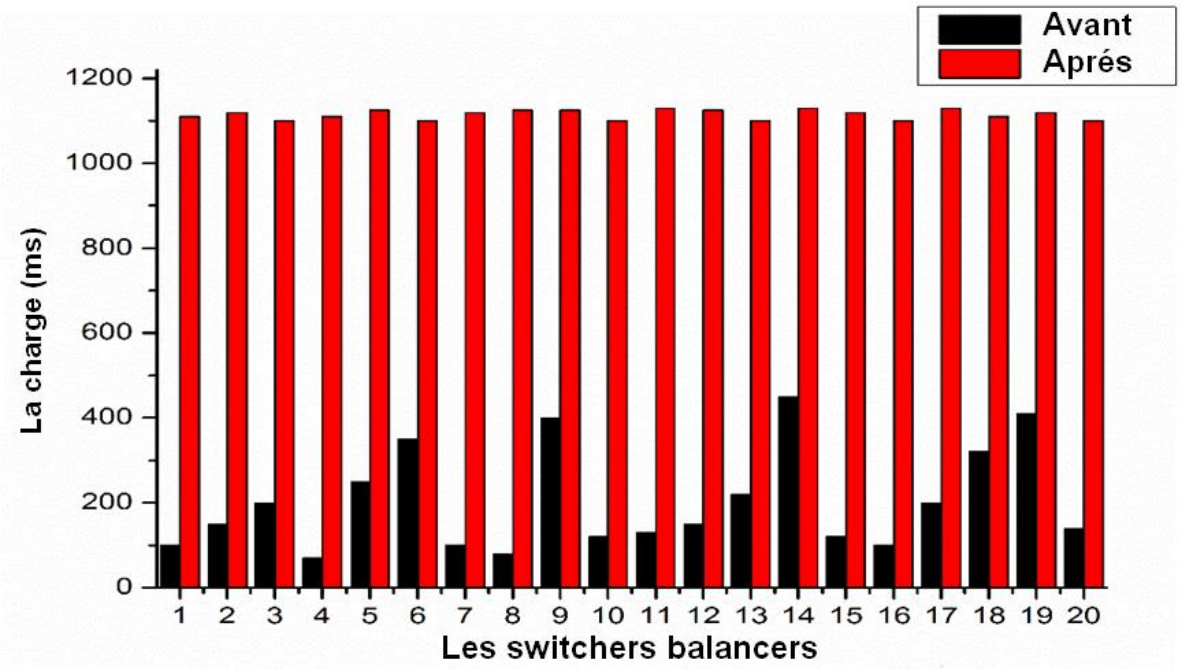

Figure 4.5 Scénario 2 : efficacité de l'algorithme 4

b) **Le deuxième scénario**

Concernant l'algorithme de manière centralisée Algorithme 2, les paramètres de ce scénario sont détaillés comme suit :

Nous fixons les ensembles de quorums à trois, le nombre de switchers balancers à chaque quorum à 20 (le total du système est de 57 switchers balancers).

Les ressources volontaires de chaque switchers balancers à 20.

Pour illustrer le graphique, nous réduisons le nombre de ressources volontaires à 20. Afin de mieux expliquer en quoi l'algorithme proposé répond aux besoins de l'utilisateur (optimisation du temps de réponse et de la disponibilité), ainsi que de la manière dont il

équilibre la charge entre les ressources de volontaires, nous n'avons pris en compte que les demandes d'un utilisateur :

- Le poids de la réponse temporelle de la demande est égal à 0,28.
- Le poids de la disponibilité est égal à 0,72.

Cette expérience concerne la charge de ressources volontaires appartenant au sous-groupe volontaire numéro 4. Chaque ressource volontaire appartenant au système se caractérise par sa probabilité d'échec, sa charge et son temps de réponse pour exécuter la demande de l'utilisateur.

Lorsque le switcher balancer reçoit la demande de l'utilisateur, comme dans cet exemple (le switcher balancer 4), il passe en deux étapes afin de trouver la solution optimale comme suit:

- ✓ ***Première étape :*** le switcher balancer calcule, dans un premier temps, la charge moyenne, puis sélectionne uniquement les ressources volontaires sous-chargées inférieures à la charge moyenne calculée. Ensuite, pour exécuter l'algorithme SA, il calcule la réponse temporelle nécessaire pour exécuter la demande de l'utilisateur par chaque ressource volontaire appartenant à la liste sous-chargée. Avant d'accéder à l'étape suivante, la normalisation de l'échelle entre le tableau de défaillances et le tableau de temps de réponse doit être satisfaite.
- ✓ ***Deuxième étape :*** le switcher balancer utilise l'algorithme de sélection (Algorithme 3) pour trouver la solution la mieux adaptée aux besoins de l'utilisateur. Comme le montre la Figure 4.6, après l'appel de la fonction de récursivité et dans l'une des situations, nous obtenons deux solutions VR1 et VR18, avec la même distance, qui est égale à 0,753, mais VR1 présente l'angle de sinus différentiel minimal par rapport aux besoins de l'utilisateur.

L'angle requis par l'utilisateur, $\sin(\alpha) = \sin(64{,}8°) = 0{,}904$ et la meilleure solution trouvée, $\sin(\beta) = \frac{0{,}66}{0{,}753} = 0{,}876$. La deuxième solution trouvée $\sin(\gamma) = \frac{0{,}36}{0{,}753} = 0{,}478$.

Alors dans cette situation, la meilleure solution avec l'angle de sinus différentiel minimal est VR1 l'angle β.

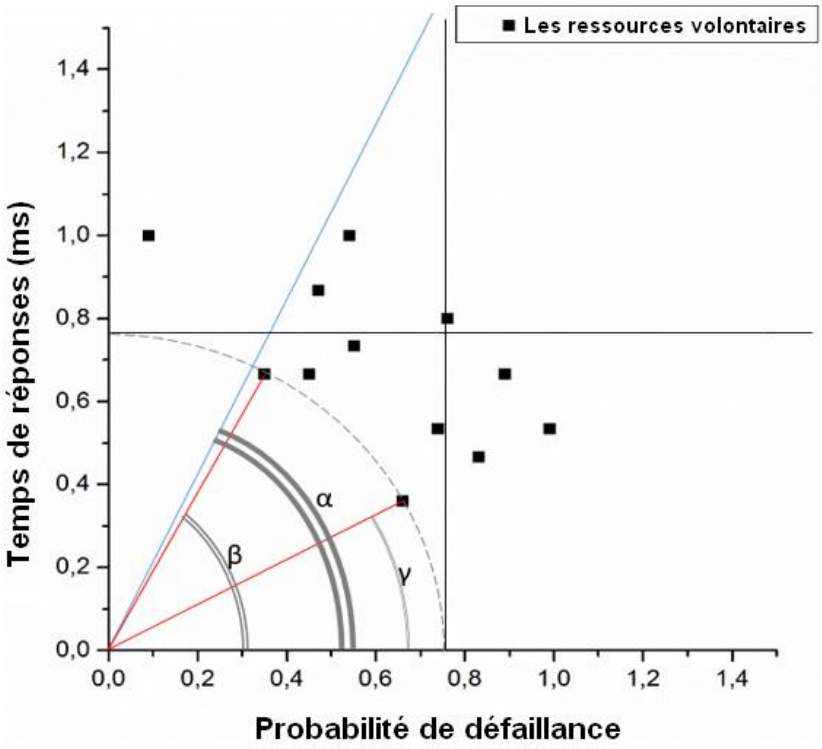

Figure 4.6 L'efficacité de l'algorithme 3

4.3.3 **Scénario 3 : Efficacité du système de quorum**

Dans ce scénario, nous comparons deux systèmes :

- Le premier utilisait le système à base de quorum et présente la configuration suivante : trois (03) quorums à chaque quorum à 35 switchers balancers, avec une propriété d'intersection entre eux, le total est de 102 switchers balancers dans tout le système.
- Le second est un système avec une architecture pure P2P lorsque nous n'utilisons aucune technique et que le nombre de switcher balancers est de 102.

Lors de la simulation, nous constatons une grande variation du nombre de messages échangés dans le système, comme montre la Figure 4.7 dans le système à base de quorum (ligne rouge), la courbe reprend progressivement. Soit dans le système avec l'architecture pure (ligne noire), la courbe augmente de façon exponentielle. À travers ce scénario, nous voulons montrer l'importance d'utiliser le système à base de quorum en réalité que l'autre.

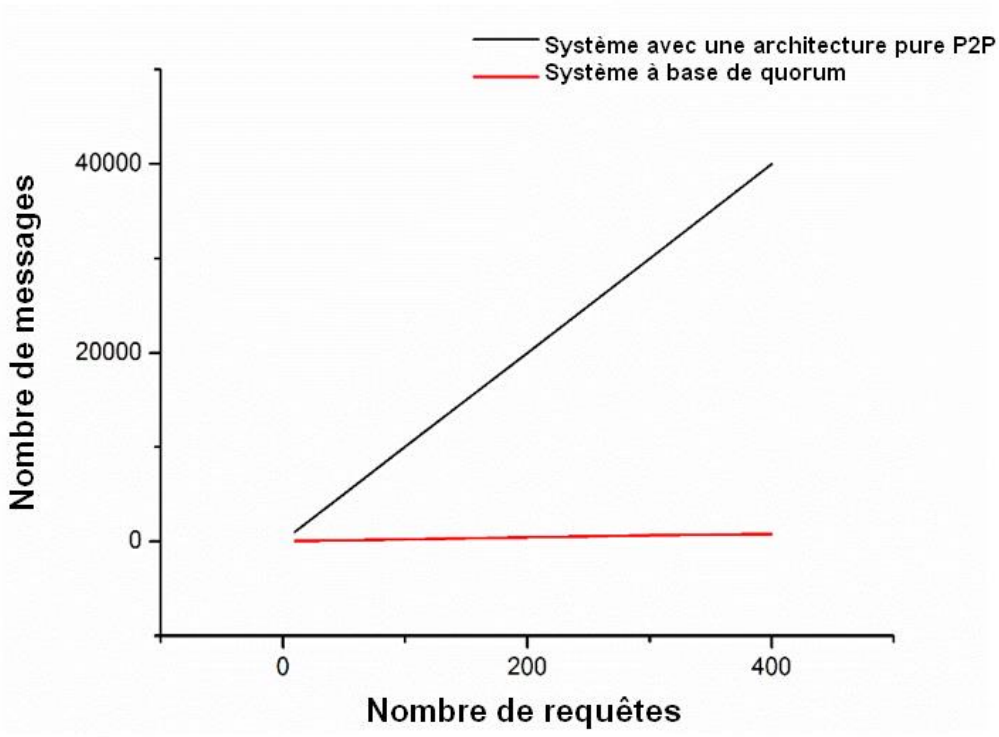

Figure 4.7 Scénario 3 : Efficacité du système de quorum.

4.3.4 Scénario 4 : analyser l'impact de l'échec des ressources volontaires sur l'équilibrage de la charge du nuage de volontaires

L'objectif de ce scénario est d'analyser l'impact de la défaillance des ressources volontaires sur l'équilibrage de charge. Pour cela, nous ne prenons que 10 switchers balancers, chacun contenant 20 ressources volontaires et chacune d'elles ayant sa propre probabilité de défaillance, ce dernier est estimé par le modèle de chaîne de Markov. Les demandes des utilisateurs étant de 2000.

Dans cette simulation, nous visons à observer comment l'algorithme de tolérance aux pannes maintient l'équilibrage de charge même après l'apparition d'une défaillance. La charge des ressources de volontaires est vérifiée avant et après la présence d'une panne sur l'un des sous-groupes volontaires, comme indiqué dans la Figure 4.8. En cas de défaillance, l'algorithme de tolérance aux pannes génère plus de messages pour acheminer les demandes en attente vers les autres ressources volontaires.

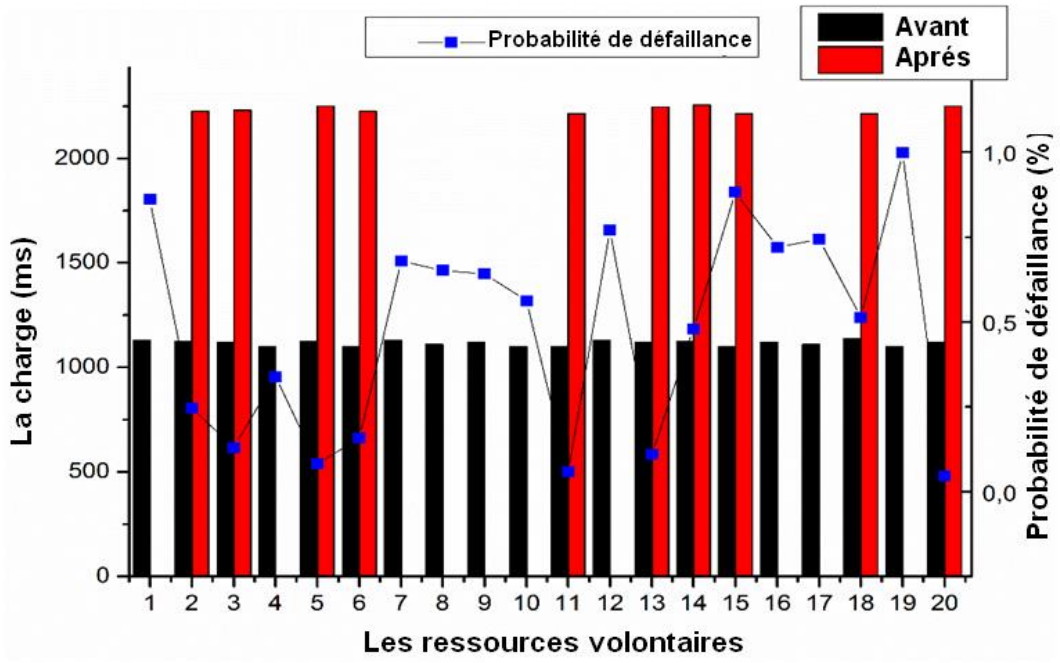

Figure 4.8 Scénario 4 : analyser l'impact de l'échec des ressources volontaires sur l'équilibrage de la charge du système.

4.3.5 Scénario 5 : L'impact de l'augmentation du nombre de quorums au nombre de messages échangés dans le système

Afin d'observer l'impact du nombre de quorums sur le nombre de messages échangés dans le système, nous fixons le nombre de demandes d'utilisateurs à 500, le nombre de ressources volontaires de chacun switcher balancer à 200, le nombre de switchers balancers à 200. Ensuite, nous faisons varier le nombre de quorums de 3 à 100 tout en préservant le même nombre total des switchers balancers.

Figure 4.9 montre que le nombre de messages correspondant à l'augmentation du nombre de quorums. Les résultats obtenus de cette expérimentation montrent que lorsque le nombre de quorums augmente, le nombre de messages augmente, c'est-à-dire que lorsque le nombre de quorums est égal à 3, le nombre de messages est égal à 1002 et lorsque le nombre de quorums devient 100, le nombre de messages atteints 1067.

Par rapport au travail de (Hemam & Hioual, 2017), nous obtenons presque les mêmes résultats, mais la seule chose ajoutée dans nos recherches est la minimisation de l'exploitation des ressources possible sans ajouter d'autres surveillances dans le système.

De cela nous concluons que le nombre de quorum utilisé doit être limité.

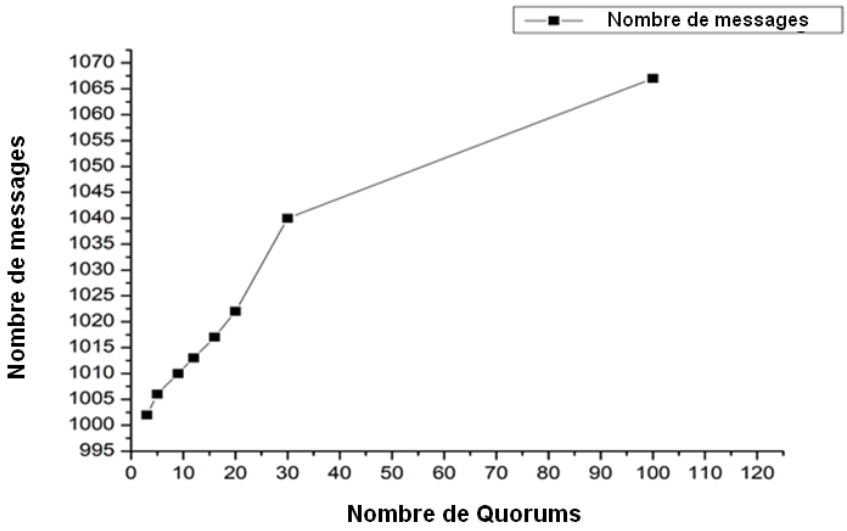

Figure 4.9 Scénario 5 : Analyser l'impact de l'augmentation du nombre de quorums sur le nombre de messages échangés dans le système.

4.4 Conclusion

Pour tester la faisabilité de notre système, nous avons implémenté nos algorithmes et notre architecture dans un environnement P2P en utilisant PeerSim comme simulateur. Nous avons étudié les résultats de simulation obtenus dans les différents scénarios en fonction de différents paramètres : nombre de demandes, l'effet de la panne, nombre de quorums.

En conclusion, les résultats obtenus montrent que notre approche qui se base sur le système de quorums et l'estimation de la défiance en utilisant les chaines de Markov, avec les deux algorithmes d'équilibrage de charge sous les contraintes des utilisateurs, permet un gain considérable en temps de réponse. Aussi, nous pouvons constater que notre approche supporte convenablement le passage à l'échelle.

Conclusion générale et perspectives

Grâce à son succès dans le paradigme des systèmes distribués dans nos jours, les systèmes des calculs volontaires attirent, de plus en plus, l'attention des chercheurs en informatique dans le monde entier.

L'un des sujets cruciaux dans les systèmes du calcul volontaire : la tolérance aux pannes, ainsi qu'il est considéré comme un thème important dans la conception de systèmes distribués. Fondamentalement, la tolérance aux pannes est connue comme étant la capacité d'un système où est capable de fonctionner en présence d'une défaillance.

1. Contribution

Les travaux présentés dans ce manuscrit s'inscrivent dans le contexte de l'équilibrage de charge et la tolérance aux pannes dans un environnement du calcul volontaire. Les problèmes traités concernant la tolérance aux pannes se concentrent beaucoup plus sur la technique de Check-pointing, qui coûte cher, et consomme plus de ressource pour sa mise en œuvre. La plupart de ces travaux comparés sont réactifs pour détecter le défaut et réparer le système (aucune prédiction des pannes).

Concernant l'équilibrage de charge, la plupart des travaux ne traitent pas l'équilibrage de charge sous les contraintes des usagers.

Nous avons proposé une architecture basée sur un système de calcul volontaires en trois couches :

1) Les sous-groupes volontaires contenaient des ressources de volontaires peu fiables.
2) Les utilisateurs envoient leurs demandes au switcher balancer en indiquant leurs besoins en matière de préférences.
3) Le système, composé d'un ensemble de switchers balancers (super-nœuds) connectés entre eux via une connexion fiable.

L'architecture proposée assure la disponibilité, la transparence, l'équilibrage de charge, la tolérance aux pannes, et le passage à l'échelle.

En résumé, cette thèse a permis :

1) La proposition d'un algorithme pour équilibrer la charge dans les sous-groupes volontaires de manière centralisée entre les ressources volontaires.

2) La proposition d'un algorithme pour équilibrer la charge dans le système entre les différents sous-groupes volontaires de manière distribuée.

3) La proposition d'un algorithme permettant de satisfaire les besoins de l'utilisateur et garde les performances du système.

4) L'utilisation du modèle de chaîne de Markov à processus stochastiques pour estimer la probabilité d'échec de chaque ressource volontaire.

Nous avons validé notre approche à travers une série de simulations en utilisant le simulateur PeerSim. Les différentes expériences ont montré que notre approche satisfait non seulement les exigences de performances des utilisateurs (temps de réponse) mais garde la performance du système (la disponibilité, la transparence, la prédiction des pannes et les traitent).

Le but de ces expériences est de tirer parti des avantages et de définir les lacunes de notre proposition.

2. Perspectives

L'architecture proposée permet d'équilibrer la charge entre les ressources volontaires, en tenant compte des contraintes imposées par les besoins des utilisateurs dans les systèmes de calculs volontaires. Dans cette architecture, les demandes sont directement envoyées au switcher balancer, ce qui nécessite l'optimisation des deux critères : le temps de réponse et la probabilité d'échec.

Dans le cadre des travaux futurs, notre objectif est de traiter la fiabilité des ressources volontaires (court terme).

En outre, nous visons un système totalement décentralisé (la plate-forme ne contienne que les ressources volontaires), (moyen terme).

Proposer un modèle pour la tolérance aux pannes (moyen terme).

La simulation sur une plateforme réelle peut gaspiller beaucoup de puissance de calcul. Un banc d'essai basé sur la simulation on des simulateurs éviterait ces problèmes. Pour cela, nous souhaiterons développer une plate-forme intégrée dans le simulateur PeerSim conviviale et réutilisable (long terme).

Production scientifique

- Revues internationales avec comité de lecture

Ledmi Abdeldjalil*, Bendjenna Hakim, and Hemmam Soufiane Mounine, Optimizing Both the User Requirements and the Load Balancing in the Volunteer Computing System by using Markov Chain Model. *International Journal of Enterprise Information Sys*tems (IJEIS), (2018), *14*(*1*), 35-62.

- Conférences internationales avec comité de lecture

Ledmi Abdeldjalil*, Bendjenna Hakim, and Hemmam Soufiane Mounine. (2018, October). Fault Tolerance in Distributed Systems: A Survey. In *2018 3rd International Conference on Pattern Analysis and Intelligent Systems (PAIS)* (pp. *1-5*). IEEE.

Annexe A : Simulation des réseaux P2P avec PeerSim

A.1 Définition

PeerSim est un simulateur Peer-to-Peer. Il a été conçu pour être dynamique et extensible. Les moteurs se composent des composants qui peuvent « être branchés » et emploient un mécanisme basé simple de configuration de fichier ASCII Que les aides réduisent les frais généraux.

La philosophie de PeerSim est d'employer une approche modulaire, comme manière préférée du codage avec lui est de réutiliser les modules existants. Ces modules peuvent être de différentes sortes, par exemple il y a des modules qui peuvent construire et initialisent le réseau fondamental, les modules qui peuvent manipuler les différents protocoles, modules pour commander et modifier le réseau. PeerSim offre beaucoup de ces modules dans ses sources, qui soulagent considérablement le codage de nouvelles applications (Legtchenko, 2008).

PeerSim peut également fonctionner en deux modes différents : cycle-basé ou événement basé.

PeerSim est sous le permis de source ouverte de GPL et est en partie développé dans Projet de BISON. PeerSim est écrit dans le langage Java et est principalement maintenu par Mark Jelasity, Gian Paolo Jesi, Alberto Montresor et Spyros Voulgaris de l'université de Bolonia (en Italie).

A.2 Fonctionnement général

Comme nous avons dit avant que le simulateur PeerSim soit basé sur plusieurs composants, qui peuvent être divisés dedans fondamentalement dans 4 classes :

- ***Protocoles (protocols)*** : ils sont habitués pour définir le comportement des différents pairs. Ils peuvent être pour des utiles différents, par exemple manipulation et simulation du réseau de substitution, ou mise en œuvre d'un algorithme distribué, comme en peut le voir dans (la Figure A.1).
- ***Nœuds (nodes)*** : ils représentent le pair eux-mêmes dans le réseau de P2P. Chaque nœud a une pile de protocole qui définira son comportement.

- ***Commandes (controls)*** : pendant que leur nom implique, les commandes peuvent commander la simulation, à intervalles réguliers ou pendant dans l'initialisation de la simulation. Elles peuvent être des observateurs simples qui recueilleront des statistiques et les imprimeront, mais ils peuvent également modifier la simulation elle-même pour changer son comportement.

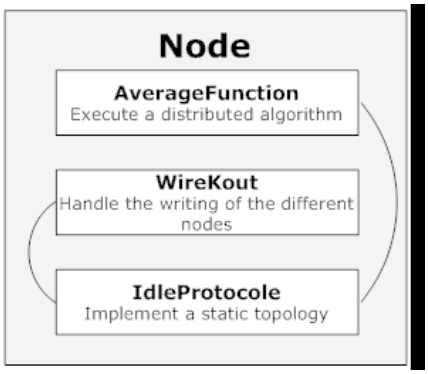

Figure A.1 Un exemple de protocole empilant sur un nœud (Legtchenko, 2008).

Réseaux (Network) : c'est une classe pour manipuler les nœuds.

A.3 Les deux modes de PeerSim

PeerSim est basé sur deux modes, qui sont très différents.

- ***Le moteur à base de cycle (cycle-based engine)*** : pendant que son nom implique, le moteur cycle-basé est basé sur des cycles. A chaque cycle, le simulateur s'attaque par chaque nœud du réseau et exécute chaque protocole associé à ce nœud. Des commandes sont également exécutées périodiquement pour commander la simulation. Elle est basée sur la classe CDSimulator du paquet de peersim.cdsim, la Figure présente comment des commandes et les protocoles sont programmés.

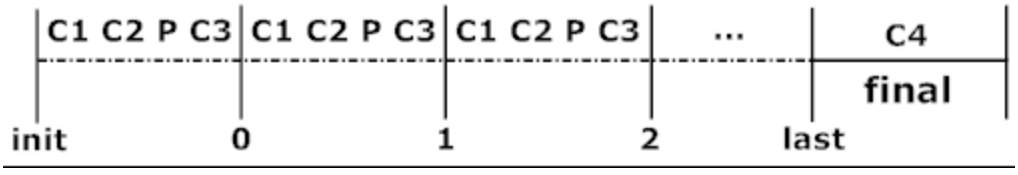

Figure A.2 Nous pouvons voir comment des commandes et les protocoles sont programmés. (Legtchenko, 2008)

- ***Le moteur basé sur les événements (event-based engine)*** : le moteur basé par événement a une manière différente de programmer des événements. Au lieu de programmer l'exécution des différents ou par les protocoles eux-mêmes), et aux

protocoles peuvent manipuler ces messages et répondre à eux en conséquence. Elle est basée sur la classe EDSimulator du paquet de peersim.edsim. En raison du fait qu'il s'appuie sur les messages, l'événementiel simulateur peut émuler une couche de transport, ajoutant ainsi plus de réalisme à la simulation. protocoles avec des cycles, ils sont programmés par des événements. Des événements (ou les messages) sont envoyés aux différents protocoles (par exemple par les composants de commande.

A.4 Le fichier de configuration

Chaque simulation est configurée à l'aide d'un fichier de configuration, qui définissent quels composants seront employés dans la simulation et comment ils agiront l'un sur l'autre les uns avec les autres. Le fichier est un fichier ASCII Simple. Chaque commentaire est mis en tête avec un # au début de la ligne. Il n'y a aucun ordre dans le fichier de configuration pour les instructions, toutefois l'exemple suivant (Figure A.3) montre comment commander les lignes un peu. Dans le fichier Figure A.3 ci-dessus nous pouvons voir la configuration de la simulation d'un établissement d'un moyen algorithme qui sera couru en mode cycle-basé pour un réseau avec 100000 nœuds (ligne 5) et pour 1000 cycles (ligne 3). D'abord nous donnons à la simulation une topologie de réseau de base avec le protocole IdleProtocol (ligne 9), excepté laquelle ne fait rien manipulant les raccordements entre les nœuds.

```
1   ## Global simulation properties
2   random.seed                 1234567890
3   simulation.cycles           1000
4   control.shf                 Shuffle
5   network.size                100000
6
7   ## Protocols
8   # Select a topology initializer
9   protocol.lnk                IdleProtocol
10  # Set the request manager protocol
11  protocol.avg                example.aggregation.AverageFunction
12  protocol.avg.linkable       lnk
13
14  ## Initialisations
15  # for the network topology
16  init.rnd                    WireKOut
17  init.rnd.protocol  lnk
18  init.rnd.k                  20
19  # distribution des valeurs sur les noeuds
20  init.peak                   example.aggregation.PeakDistributionInitializer
21  init.peak.value             10000
22  init.peak.protocol          avg
23
24  ## Controls
25  control.avgo                example.aggregation.AverageObserver
26  control.avgo.protocol       avg
```

Figure A.3 Un fichier de configuration de base.

Sur chaque nœud et à chaque cycle, un algorithme mis en application par Average-Function (ligne 11), calculeront le moyen entre la valeur contenue sur celle des nœuds et un de son voisin. L'objectif de cet algorithme est l'établissement d'une moyenne distribuée . AverageFunction est réellement une classe de PeerSim qui peut être trouvée dans l'annuaire /example/Aggregation.

Sur la ligne 14 les composants qui initialisent la simulation sont définis. WireKOut à la ligne 16 manipule le câblage des nœuds et le degré du graphique (ligne 18). Puis les valeurs sur différents nœuds sont initialisées avec le PeakDistributionInitializer composant (ligne 20), qui donnera une valeur de 10000 sur un nœud, et 0 et les autres (ligne 21). En conclusion, nous ajoutons une commande à la simulation, AverageObserver (ligne 25) qui recueillera les statistiques et les imprimera régulièrement.

Les résultats AverageObserver nous a laissés imprimer le rendement suivant : (Figure A.4)

```
1  Simulator: loading configuration
2  ConfigProperties: File example/config-test.txt loaded.
3  Simulator: starting experiment 0 invoking peersim.cdsim.CDSimulator
4  Random seed: 1234567890
5
6  CDSimulator: resetting
7  Network: no node defined, using GeneralNode
8  CDSimulator: running initializers
9  - Running initializer init.peak: class example.aggregation.PeakDistributionInitializer
10 - Running initializer init.rnd: class peersim.dynamics.WireKOut
11 CDSimulator: loaded controls [control.avgo, control.shf]
12 CDSimulator: starting simulation
13 control.avgo: 0 0.0 10000.0 100000 0.1 1000.0 99999 1
14 CDSimulator: cycle 0 done
15 control.avgo: 1 0.0 2500.0 100000 0.1 187.49187491874918 99994 2
16 CDSimulator: cycle 1 done
17 control.avgo: 2 0.0 1250.0 100000 0.1 54.06768911439114 99972 1
18 CDSimulator: cycle 2 done
19 control.avgo: 3 0.0 625.0 100000 0.1 19.366948430226486 99848 2
20 CDSimulator: cycle 3 done
21 control.avgo: 4 0.0 312.5 100000 0.1 6.383266107582426 99206 2
22 CDSimulator: cycle 4 done
23 control.avgo: 5 0.0 156.25 100000 0.1 2.0482095720890525 95974 1
24 CDSimulator: cycle 5 done
25 control.avgo: 6 0.0 78.277587890625 100000 0.1 0.75166241219096 81391 2
26 CDSimulator: cycle 6 done
27 control.avgo: 7 0.0 39.1387939453125 100000 0.1 0.239095922144 39078 2
28 CDSimulator: cycle 7 done
29 control.avgo: 8 0.0 19.56939697265625 100000 0.1 0.070958498117 3906 1
30 CDSimulator: cycle 8 done
31 ...
```

Figure A.4 Premières lignes du résultat de la simulation.

D'abord nous avons l'information de base comme le nom du fichier de configuration (ligne 2), la graine pour produire de nombres aléatoires (ligne 4), le simulateur utilisé (CDSimulator cycle basé, ligne 3) et les initialiseurs (ligne 9 et 10).

Alors nous avons les résultats de la simulation eux-mêmes. Seulement les 8 premiers cycles (sur 10000) sont montrés. Aux lignes 13, 15, 17, 19 et ainsi de suite nous pouvons voir le rendement du AverageObserver (avgo), la commande que nous avons vue précédemment dans le fichier de configuration.

A.5 Simulation événementielle

Toutes les plates-formes de simulation sont basées sur le modèle à événements discrets. Idée : Discrétiser le temps, la Figure A.5 présente l'évolution du temps global au sein de la simulation en fonction des messages reçus.

- Deux entités : les nœuds et les messages.
- On considère que le temps évolue seulement lorsqu'un événement survient sur un nœud.

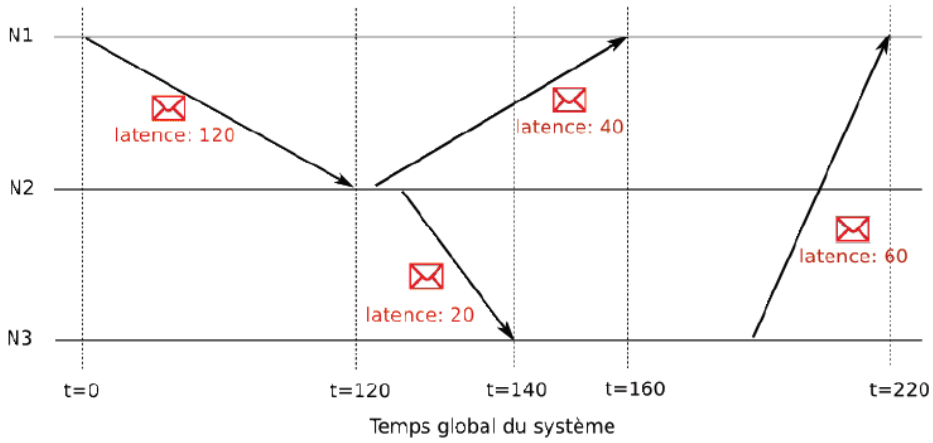

Figure A.5 Evolution du temps global au sein de la simulation en fonction des messages (Legtchenko, 2008)

A.5.1 Gestion des messages

Chaque message envoyé est estampillé avec son temps de réception.

- Lors de l'envoi, les messages sont insérés dans une fille à priorités en fonction de leur estampille.

- Lorsque le simulateur a calculé l'état de l'application à un instant, il récupéré un nouveau message en tête de file, et le délivre au nœud destinataire.

On simule en fait la réaction du système aux événements :

- Réception de messages.
- Evènements internes aux nœuds.

Mais pas le comportement du système entre les événements.

A.5.2 **Avantages**

✓ La charge de calcul est allégée : on ne simule pas le comportement du système entre les évènements.

✓ Il est possible de faire des sauvegardes d'un état du système, il est donc très simple de faire des arrêts du système pour reprendre la simulation à un autre moment.

✓ La simulation est reproductible : on peut redéclencher le même bug encore et encore jusqu'à sa résolution.

✓ Le debug est facilité : on connait à tout moment les messages en transit et leur temps de délivrèrent

✓ On ne code que les traitants des évènements

A.5.3 **Inconvénients**

On fait des hypothèses de simplification par rapport au système réel Il faut correctement choisir le grain des événements :

✗ Un grain trop fin ralentit excessivement la simulation

✗ Un grain trop gros risque de fausser le résultat de la simulation

On peut perdre du temps par rapport à la réalité si le système qu'on simule génère trop d'événements.

A.6 **Entités principales**

La Figure A.6 présente le fonctionnement général du PeerSim.

❖ Node (classe) : représente un nœud.

❖ EDProtocol (interface) : représente un protocole.

- Network (classe statique) : classe pour manipuler les nœuds.
- EDSimulator (classe statique) : gère les événements.

A.6.1 Classe Network

Il s'agit d'un tableau qui contient L'ensemble des nœuds

Méthodes utiles

- Network.add : ajoute un nœud.
- Network.get : retourne le nœud dont l'index est en paramètre.
- Network.remove : supprime un nœud.

A.6.2 Classe Node

Chaque nœud contient une instance de chaque protocole défini, comme la montre (les figures Figure A.6, Figure A.7).

Méthodes utiles

- setFailState : Activer/Désactiver un nœud (simuler une connexion/déconnexion).
- getFailState : Information sur l'état du nœud (activé ou désactivé).
- getProtocol : retourne l'objet protocole dont l'identifiant est passé en paramètre.

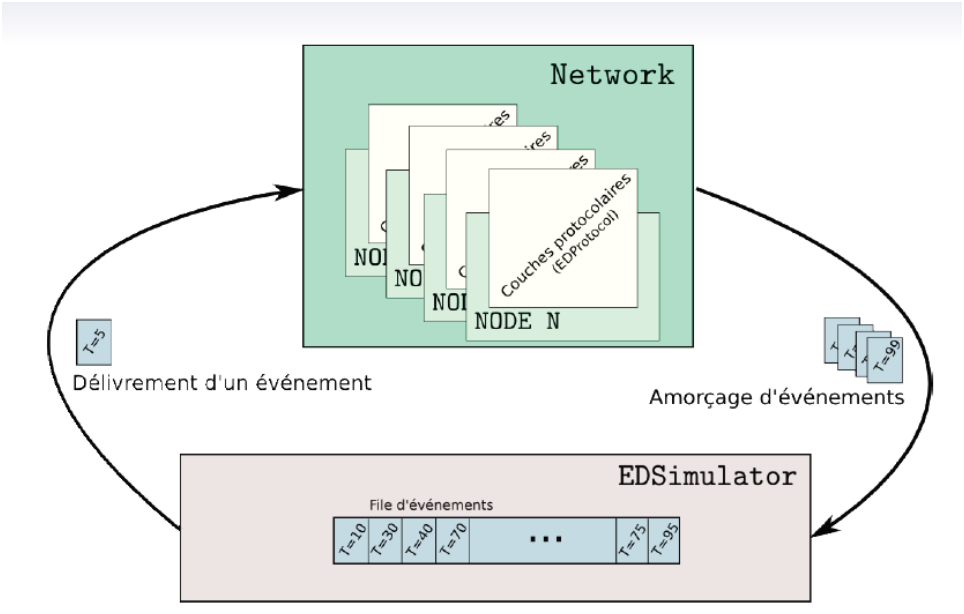

Figure A.6 Vue d'ensemble du simulateur Peersim (Legtchenko, 2008)

A.6.3 Interface EDProtocol

- Symbolise une couche réseau.
- Un objet avec l'interface EDProtocol est associé à un identifiant de protocole.
- Deux méthodes obligatoires : processEvent et clone.

A.6.4 Classe EDSimulator

- Contient la file d'événements
- Extrait les événements en tête de file et les délivre aux nœuds destinataires

Méthodes utiles

- setFailState : Activer/Désactiver un nœud (simuler une connexion/déconnexion)
- getFailState : Information sur l'état du nœud (activé ou désactivé)
- getProtocol : retourne l'objet protocole dont l'identifiant est passé en paramètre.

Envoi d'un message On invoque EDSimulator.add (t,event,N,pid) ; Le message mis en file d'attente possède donc :

- t : Un temps de délivrèrent
- N : Un nœud cible sur lequel délivrer le message
- pid : L'identifiant d'un protocole

Au temps t, la méthode processEvent du protocole ayant l'identifiant pid situé sur le nœud N sera invoquée par le simulateur.

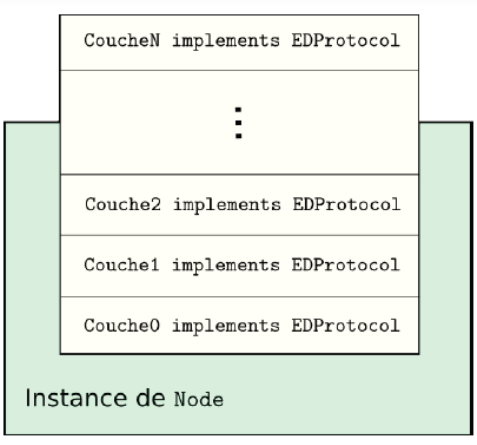

Figure A.7 Chaque objet Node contient une instance de chaque protocole. Il s'agit simplement (Legtchenko, 2008)

A.6.5 Modules d'initialisation

- Classe implémentant l'interface peersim.core.Control.
- Possède une méthode exécute invoquée une seule fois au début de la simulation.

Nécessaire pour le bootstrap du système

- Construction de la topologie du système.
- Amorçage de la couche applicative.

A.6.6 Modules de contrôle

- Classe implémentant l'interface peersim.core.Control.
- Sa méthode exécute est périodiquement invoquée pendant toute la durée de la simulation.

Permet de simuler

- Des événements périodiques du système (protocoles de maintenance).
- L'activité de la couche applicative.
- Des événements extérieurs : pannes, départs de nœud s, etc...

Diagramme type d'exécution de la simulation, comme la montre (la figure A.8) après la phase d'initialisation, les modules de contrôle sont exécutés de façon périodique.

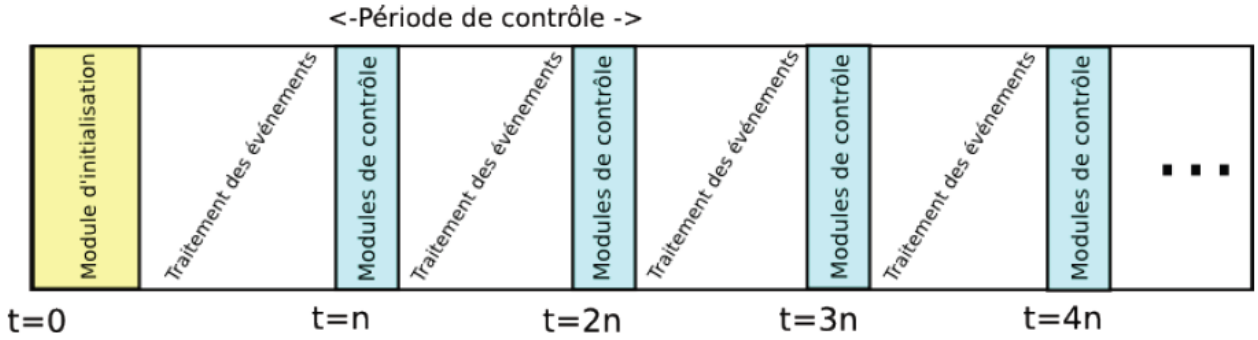

Figure A.8 la phase d'initialisation, les modules de contrôle sont exécutés de façon (Legtchenko, 2008)

A.7 Spécification des couches protocolaires

Soit un modèle avec deux couches, comme la montre (la Figure A.9) :

- Couche transport : la classe MyTransport
- Couche applicative : la classe MyApplicative
- La couche applicative utilise la couche transport pour propager ses messages.
- Une instance de MyApplicative doit donc posséder une instance de MyTransport.

Comment faire ?

- On peut créer un objet MyTransport à l'initialisation de MyApplicative. Problème : Si on veut remplacer MyTransport par MyTransport2, il faut changer le code de MyApplicative.
- On peut spéciér la dépendance entre MyApplicative et MyTransport dans le fichier de configuration.

On conserve ainsi la généricité des couches protocolaires.

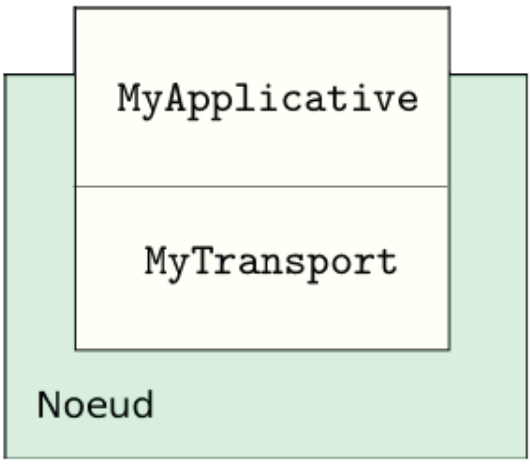

Figure A.9 Les deux couches protocolaires
(Legtchenko, 2008).

A.8 Spécification d'un module d'initialisation

Soit une classe MyBootstrap initialisant le système. Dans le fichier de configuration : init.mondemarrage

- Le mot clé init précise qu'il s'agit d'un module d'initialisation.
- mondemarrage est le nom de l'entrée.
- MyBootstrap est la valeur de l'entrée (ie. Le nom de la classe).

- Au lancement du simulateur, un chargeur de classe est invoqué pour charger les classes spécifiées dans le fichier de configuration.

A.9 Mots clés importants du fichier de configuration

- init.monmodule MaClasse charge MaClasse en tant que module d'initialisation.
- control.monmodule MaClasse charge MaClasse en tant que module de contrôle.
- protocol.monmodule MaClasse charge MaClasse en tant que protocole.
- simulation.experiments permet de spécifier le nombre d'expériences consécutives (en général une seule).
- simulation.endtime permet de spécifier le temps de terminaison de l'expérience.
- network.size spécifié la taille du réseau (en nombre de nœuds).

Bibliographie

Abramson, D., Buyya, R., & Giddy, J. (2002). A computational economy for grid computing and its implementation in the Nimrod-G resource broker. *Future Generation Computer Systems, 18*(8), 1061-1074.

Aguilera, M. K., Chen, W., & Toueg, S. (1999). Using the heartbeat failure detector for quiescent reliable communication and consensus in partitionable networks. *Theoretical Computer Science, 220*(1), 3-30.

Ali, M. F., & Khan, R. Z. (2012). The study on load balancing strategies in distributed computing system. *International Journal of Computer Science Engineering Survey, 3*(2), 19.

Amazon. Amazon Elastic Compute Cloud (Amazon EC2). . Retrieved from http://aws.amazon.com/

Anderson, D. P. (2004). *BOINC: A System for Public-Resource Computing and Storage," 5th IEEE.* Paper presented at the ACM International Workshop on Grid Computing, Pittsburgh, PA (November 8, 2004). See also http://boinc. berkeley. edu.

Anderson, D. P., Cobb, J., Korpela, E., Lebofsky, M., & Werthimer, D. (2002). SETI@ home: an experiment in public-resource computing. *Communications of the ACM, 45*(11), 56-61.

Anderson, D. P., & McLeod, J. (2007). *Local scheduling for volunteer computing.* Paper presented at the 2007 IEEE International Parallel and Distributed Processing Symposium.

Antoniu, G., Deverge, J. F., & Monnet, S. (2006). How to bring together fault tolerance and data consistency to enable grid data sharing. *Concurrency Computation: Practice Experience, 18*(13), 1705-1723.

Avizienis, A., Laprie, J.-C., Randell, B., & Landwehr, C. (2004). Basic concepts and taxonomy of dependable and secure computing. *IEEE transactions on dependable secure computing, 1*(1), 11-33.

Bala, A., & Chana, I. (2015). Autonomic fault tolerant scheduling approach for scientific workflows in Cloud computing. *Concurrent Engineering, 23*(1), 27-39.

Banerjee, S., & Hecker, J. P. (2017). *A Multi-agent system approach to load-balancing and resource allocation for distributed computing.* Paper presented at the First Complex Systems Digital Campus World E-Conference 2015.

Barrera, H. E. C., Rosero, E. E. R., & Cano, M. J. V. (2012a). Desktop grids and volunteer computing systems. In (pp. 1-30): IGI Global.

Barrera, H. E. C., Rosero, E. E. R., & Cano, M. J. V. (2012b). Desktop grids and volunteer computing systems. In *Computational and Data Grids: Principles, Applications and Design* (pp. 1-30): IGI Global.

Behera, I., & Tripathy, C. R. (2014). Performance modelling and analysis of mobile grid computing systems. *International Journal of Grid Utility Computing, 5*(1), 11-20.

Bernard, G., Stève, D., & Simatic, M. (1993). A survey of load sharing in networks of workstations. *Distributed Systems Engineering, 1*(2), 75.

BOINC. (2018). Retrieved from https://boinc.berkeley.edu/

Brasileiro, F., & Miranda, R. (2009). *The ourgrid approach for opportunistic grid computing.* Paper presented at the Proceedings of the First EELA-2 Conference.

Castro, H., Rosales, E., Villamizar, M., & Jiménez, A. (2010). *Unagrid: On demand opportunistic desktop grid.* Paper presented at the Proceedings of the 2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing.

Cirne, W., Brasileiro, F., Sauve, J., Andrade, N., Paranhos, D., Santos-neto, E., & Medeiros, R. (2003). *Grid computing for bag of tasks applications.* Paper presented at the In Proc. of the 3rd IFIP Conference on E-Commerce, E-Business and EGovernment.

Clarke, I., Sandberg, O., Wiley, B., & Hong, T. W. (2001). *Freenet: A distributed anonymous information storage and retrieval system.* Paper presented at the Designing privacy enhancing technologies.

Coulouris, G. (2011). Distributed systems concepts and design. In: Boston: Addison-Wesley.

Cumulus Project. Retrieved from http://www.cumulus-project.eu/

Dabrowski, C., & Hunt, F. (2009). *Using markov chain analysis to study dynamic behaviour in large-scale grid systems.* Paper presented at the Proceedings of the Seventh Australasian Symposium on Grid Computing and e-Research-Volume 99.

Diekmann, R., Frommer, A., & Monien, B. (1999). Efficient schemes for nearest neighbor load balancing. *Parallel Computing, 25*(7), 789-812.

Distributed.Net. (1997). Retrieved from https://www.distributed.net/Main_Page

Dong, F., & Akl, S. G. (2006). *Scheduling algorithms for grid computing: State of the art and open problems*. Retrieved from

Eberspächer, J., & Schollmeier, R. (2005). 5. first and second generation of peer-to-peer systems. In *Peer-to-Peer Systems and Applications* (pp. 35-56): Springer.

El-Zoghdy, S., & Ghoniemy, S. (2014a). A Survey of Load Balancing In High-Performance Distributed Computing Systems. *J International Journal of Advanced Computing Research, 1.*

El-Zoghdy, S., & Ghoniemy, S. (2014b). A Survey of Load Balancing In High-Performance Distributed Computing Systems. *International Journal of Advanced Computing Research, 1.*

El-Zoghdy, S. F. (2012). A CAPACITY-BASED LOAD BALANCING AND JOBMIGRATION ALGORITHM FOR HETEROGENEOUS COMPUTATIONAL GRIDS. *International Journal of Computer Networks Communications, 4*(1), 113.

El-Zoghdy, S. F., & Elnashar, A. I. (2015). A threshold-based load balancing algorithm for grid computing systems. *Journal of High Speed Networks, 21*(4), 237-257.

Elnozahy, E. N., Alvisi, L., Wang, Y.-M., & Johnson, D. B. (2002). A survey of rollback-recovery protocols in message-passing systems. *ACM Computing Surveys, 34*(3), 375-408.

Eucalyptus. Retrieved from. https://www.eucalyptus.com/

Facebook. Retrieved from http://www.facebook.com

Folding@home. Retrieved from http://folding.stanford.edu.

Frey, J., Tannenbaum, T., Livny, M., Foster, I., & Tuecke, S. (2002). Condor-G: A computation management agent for multi-institutional grids. *Cluster Computing, 5*(3), 237-246.

Ghafarian, T., Deldari, H., Javadi, B., & Buyya, R. (2013). A proximity-aware load balancing in peer-to-peer-based volunteer computing systems. *The Journal of Supercomputing, 65*(2), 797-822.

Ghomi, E. J., Rahmani, A. M., & Qader, N. N. (2017). Load-balancing algorithms in cloud computing: a survey. *Journal of Network Computer Applications, 88*, 50-71.

Giacobini, M., Tomassini, M., & Tettamanzi, A. (2005). *Takeover time curves in random and small-world structured populations.* Paper presented at the Proceedings of the 7th annual conference on Genetic and evolutionary computation.

The Gnutella Developer Forum (GDF). (2001). *The Gnutella v0.6 protocol.*

GoGrid. Retrieved from http://www.gogrid.com/.

Goldchleger, A., Kon, F., Goldman, A., Finger, M., & Bezerra, G. C. (2004). InteGrade: object-oriented Grid middleware leveraging the idle computing power of desktop machines. *Concurrency and Computation: Practice and Experience, 16*(5), 449-459.

Google Gmail. Retrieved from https://www.gmail.com/

Google. Google App Engine: Platform as a Service — Google Cloud Platform. . Retrieved from https://cloud.google.com/appengine/docs

Grid'5000. Retrieved from https://www.grid5000.fr

Hemam, S. M., & Hioual, O. (2017). A hybrid load balancing algorithm for P2P-cloud system aware of constraints optimisation of cost and reliability criteria. *International Journal of Internet Protocol Technology, 10*(2), 99-114.

Htcondor. Retrieved from http://research.cs.wisc.edu/htcondor/

Hussain, H., Malik, S. U. R., Hameed, A., Khan, S. U., Bickler, G., Min-Allah, N., . . . Ghani, N. (2013). A survey on resource allocation in high performance distributed computing systems. *Parallel Computing, 39*(11), 709-736.

Hwang, K., Dongarra, J., & Fox, G. C. (2013). *Distributed and cloud computing: from parallel processing to the internet of things*: Morgan Kaufmann.

Jain, P., & Gupta, D. (2009). An algorithm for dynamic load balancing in distributed systems with multiple supporting nodes by exploiting the interrupt service. *International Journal of Recent Trends in Engineering, 1*(1), 232.

Javadi, B., Matawie, K., & Anderson, D. P. (2013). *Modeling and analysis of resources availability in volunteer computing systems.* Paper presented at the Performance Computing and Communications Conference (IPCCC), 2013 IEEE 32nd International.

JavedHussain, & DurgeshKumar, M. (2013). Analysis and Investigation of Nearest Neighbor Algorithm for Load Balancing. *International Journal of Emerging Trends & Technology in Computer Science (IJETTCS), 2*(6).

Jelasity, M., & Steen, M. v. (2002). *Large-scale newscast computing on the Internet.* Retrieved from

Jelasity, M., Voulgaris, S., Guerraoui, R., Kermarrec, A.-M., & Van Steen, M. (2007). Gossip-based peer sampling. *ACM Transactions on Computer Systems, 25*(3), 8.

Kademlia, A. (2001). A peer-to-peer information system based on the xor metric. *Webpage available at:, published Mar.*

Kameda, H., Li, J., Kim, C., & Zhang, Y. (2012). *Optimal load balancing in distributed computer systems*: Springer Science & Business Media.

Khan, Z., Singh, R., Alam, J., & Saxena, S. (2011). Classification of load balancing conditions for parallel and distributed systems. *International Journal of Computer Science Issues, 8*(5), 411.

Kirk, P. (2003). *The Annotated Gnutella Protocol Specification v0. 4.* Paper presented at the The Gnutella Developer Forum (GDF).

Lamport, L. (1987). Distribution. In.

Lázaro, D., Kondo, D., & Marquès, J. M. (2012). Long-term availability prediction for groups of volunteer resources. *Journal of Parallel Distributed Computing, 72*(2), 281-296.

Ledmi, A., Bendjenna, H., & Mounine, H. S. (2018). Optimizing Both the User Requirements and the Load Balancing in the Volunteer Computing System by using Markov Chain Model. *International Journal of Enterprise Information Systems, 14*(1), 35-62.

Legtchenko, S. (2008). *Evaluation de systemes repartis a large echelle.* Retrieved from https://pages.lip6.fr/Sergey.Legtchenko/enseignements/coursPSIA.pdf

Litzkow, M. J., Livny, M., & Mutka, M. W. (1988). *Condor-a hunter of idle workstations.* Paper presented at the Distributed Computing Systems, 1988., 8th International Conference on.

Livny, M., & Raman, R. (1998). High-throughput resource management.[in:] I. Foster, C. Kesselman, eds., The Grid: Blueprint for a New Computing Infrastructure. In: Morgan Kaufmann.

Lua, E. K., Crowcroft, J., Pias, M., Sharma, R., & Lim, S. (2005). A survey and comparison of peer-to-peer overlay network schemes. *IEEE Communications Surveys Tutorials, 7*(2), 72-93.

Luke, S., Balan, G. C., & Panait, L. (2003). *Population implosion in genetic programming.* Paper presented at the Genetic and Evolutionary Computation Conference.

Malkhi, D., Naor, M., & Ratajczak, D. (2002). *Viceroy: A scalable and dynamic emulation of the butterfly.* Paper presented at the Proceedings of the twenty-first annual symposium on Principles of distributed computing.

Manekar, A. S., Poundekar, M., Gupta, H., & Nagle, M. (2012). A Pragmatic Study and Analysis of Load Balancing Techniques In Parallel Computing. *International Journal of Engineering Research Applications, 2*(4).

Mersenne, Research Inc. (1996). Retrieved from https://www.mersenne.org/

Microsoft. Microsfot Azure. Retrieved from http://azure.microsoft.com/

Mok, R. K., Bajpai, V., Dhamdhere, A., & Claffy, K. (2018). *Revealing the Load-Balancing Behavior of YouTube Traffic on Interdomain Links.* Paper presented at the International Conference on Passive and Active Network Measurement.

Mokadem, R., Hameurlain, A., & Tjoa, A. M. (2012). Resource discovery service while minimizing maintenance overhead in hierarchical DHT systems. *International Journal of Adaptive, Resilient Autonomic Systems, 3*(2), 1-17.

Montresor, A., & Jelasity, M. (2009). *PeerSim: A scalable P2P simulator.* Paper presented at the Peer-to-Peer Computing, 2009. P2P'09. IEEE Ninth International Conference on.

Napster. Retrieved from http://www.napster.com/

Nimbus. Retrieved from http://www.nimbusproject.org

OpenNebula. Retrieved from http://opennebula.org/

PeerSim. (03/11/2018). Retrieved from https://sourceforge.net/projects/peersim/

Pitoura, T., Ntarmos, N., & Triantafillou, P. (2012). Saturn: range queries, load balancing and fault tolerance in DHT data systems. *IEEE Transactions on Knowledge Data Engineering, 24*(7), 1313-1327.

Rackspace managed cloud. Retrieved from http://www.rackspace.com/cloud

Rajguru, A. A., & Apte, S. (2012). A comparative performance analysis of load balancing algorithms in distributed system using qualitative parameters. *International Journal of Recent Technology Engineering, 1*(3), 175-179.

Ratnasamy, S., Francis, P., Handley, M., Karp, R., & Shenker, S. (2001). *A scalable content-addressable network* (Vol. 31): ACM.

Rowstron, A., & Druschel, P. (2001). *Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems.* Paper presented at the IFIP/ACM International Conference on Distributed Systems Platforms and Open Distributed Processing.

Saini, R., & Prakash, P. (2013). A New Approach to Volunteer Cloud Computing. *IOSR Journal of Computer Engineering e-ISSN*, 2278-0661.

Salehi, M. A., Deldari, H., & Dorri, B. M. (2009). Balancing load in a computational grid applying adaptive, intelligent colonies of ants. *Informatica, 33*(2).

Salesfoce Platform. Retrieved from http://www.salesforce.com

Salesforce.com Inc. Salesfoce Platform. Retrieved from http://www.salesforce.com

Sarmenta, L. F., Chua, S. J., Echevarria, P., Mendoza, J. M., Santos, R.-R., Tan, S., & Lozada, R. (2002). *Bayanihan Computing. NET: Grid Computing with XML Web Services.* Paper presented at the ccgrid.

Savio, S. (2009). Online Bicriteria Load Balancing Using Object Reallocation. *IEEE Trans. Parallel Distrib. Syst, 20*(3), 379-388.

Sharma, G. (2013). A Review on Different Approaches for Load Balancing in Computational Grid. *Journal of Global Research in Computer Science, 4*(4), 82-85.

Sharma, S., Singh, S., & Sharma, M. (2008). Performance analysis of load balancing algorithms. *World Academy of Science, Engineering Technology, 38*(3), 269-272.

Sheikhalishahi, M., Devare, M., Grandinetti, L., & Incutti, M. C. (2012). A Complementary Approach to Grid and Cloud Distributed Computing Paradigms. In (pp. 31-44): IGI Global.

Shoch, J. F., & Hupp, J. A. (1982). The "worm" programs—early experience with a distributed computation. *Communications of the ACM, 25*(3), 172-180.

Steinmetz, R., & Wehrle, K. (2005). *Peer-to-peer systems and applications* (Vol. 3485): Springer.

Stoica, I., Morris, R., Liben-Nowell, D., Karger, D. R., Kaashoek, M. F., Dabek, F., & Balakrishnan, H. (2003). Chord: a scalable peer-to-peer lookup protocol for internet applications. *IEEE/ACM Transactions on Networking, 11*(1), 17-32.

Sun Microsystems Inc. (2007). Retrieved from http://www.virtualbox.org/

Tanenbaum, A. S., & Steen, M. V. (2017). *Distributed Systems: Principles and Paradigms.*

Tanenbaum, A. S., & Van Steen, M. (2007). *Distributed systems: principles and paradigms*: Prentice-Hall.

Tijms, H. C. (2003). *A first course in stochastic models*: John Wiley and sons.

Tse, S. S. (2013). Online balancing two independent criteria upon placements and deletions. *IEEE Transactions on Parallel Distributed Systems*(8), 1644-1650.

Van Steen, M. (2010). Graph theory and complex networks. *An introduction, 144.*

Varrette, S., Plugaru, V., Guzek, M., Besseron, X., & Bouvry, P. (2014). *Hpc performance and energy-efficiency of the openstack cloud middleware.* Paper presented at the Parallel Processing Workshops (ICCPW), 2014 43rd International Conference on.

VMware Inc. (2008). Retrieved from http://www.vmware.com/

Wolski, R., Plank, J. S., Brevik, J., & Bryan, T. (2001). Analyzing market-based resource allocation strategies for the computational grid. *The International Journal of High Performance Computing Applications, 15*(3), 258-281.

XSEDE. Retrieved from https://www.xsede.org/

Zhao, B. Y., Huang, L., Stribling, J., Rhea, S. C., Joseph, A. D., & Kubiatowicz, J. D. (2004). Tapestry: A resilient global-scale overlay for service deployment. *IEEE Journal on selected areas in communications, 22*(1), 41-53.

Acronymes

BOINC	**B**erkeley **O**pen **I**nfrastructure for **N**etwork **C**omputing
DGVCS	**D**esktop **G**rids and **V**olunteer **C**omputing **S**ystems
FLOPS	**FL**oating point **O**perations **P**er **S**econd
LAN	**L**ocal **A**rea **N**etwork
PARC	**P**alo **A**lto **R**esearch **C**enter
DNS	**D**omain **N**ame **S**ystem
GIMPS	**G**reat **I**nternet **M**ersenne **P**rime **S**earch
OGR	**O**ptimal **G**olomb **R**ulers
BoT	**B**ag-**o**f-**T**asks
QOS	**Q**uality **O**f **S**ervice
COMIT	**Com**munications and **I**nformation **T**echnology Group
NAT	**N**etwork **A**ddress **T**ranslation
ISP	**I**nternet **S**ervice **P**rovider
RRA	**R**ound **R**obin **A**lgorithm
CMA	**C**entral **M**anager **A**lgorithm
TA	**T**hreshold **A**lgorithm
FIFO	**F**irst-**I**n **F**irst-**O**ut
DTMC	**D**iscrete **T**ime **M**arkov **C**hain