UNIVERSITÉ DE GRENOBLE

THÈSE

Pour obtenir le grade de

DOCTEUR DE L'UNIVERSITÉ DE GRENOBLE

Spécialité : **Informatique**

Arrêté ministérial :

Présentée par

Ahmed HARBAOUI

Thèse dirigée par **Brigitte Plateau, Professeur, ENSIMAG, Grenoble**

préparée au sein **Laboratoire Informatique de Grenoble Orange labs**

Vers une modélisation et un dimensionnement automatiques des applications réparties

Thèse soutenue publiquement le
devant le jury composé de :

Nihal Pekergin Professeur, Université Paris XII, Paris, Rapporteur
Pierre Sens Professeur, Université Paris VI, Paris, Rapporteur
Patrice Moreaux Professeur, Polytech'Savoie, Annecy, Examinateur
Thomas Begin MdC, Université Claude Bernard, Lyon, Examinateur
Martin Quinson MdC, ESIAL, Nancy, Examinateur
Brigitte Plateau Professeur, ENSIMAG, Grenoble, Directeur de thèse
Jean-Marc Vincent MdC, Université Joseph Fourier, Co-Directeur de thèse
Bruno Dillenseger Ing. recherche, Orange Labs, Co-Directeur de thèse

À ma mère,
À mon père,
À mon épouse,
À ma petite fille Eya

Remerciements

Tout d'abord, je voudrais remercier infiniment Monsieur Pierre *Sens*, professeur à l'Université Paris VI, et Madame Nihal *Pekergin*, professeur à l'Université Paris XII, d'avoir rapporté la thèse. Je leur suis très reconnaissant de l'intérêt qu'ils avaient porté à ces travaux de recherche, ceci tout en ayant un regard critique et constructif. Je tiens donc à leur exprimer ma profonde gratitude. Mes remerciements vont également à Monsieur Thomas *Begin*, maître de conférences à l'université Claude Bernard de Lyon, pour l'intérêt qu'il a manifesté à l'égard de cette thèse et d'avoir apporté des remarques pertinentes à mon travail. Je remercie également Monsieur Patrice *Moreaux*, professeur à Polytech'Savoie d'Annecy, d'avoir bien voulu présider le jury de thèse et examiner mon manuscrit, et à Monsieur Martin *Quinson*, maître de conférences à l'Ecole Supérieure d'Informatique et Applications de Lorraine de Nancy, d'avoir accepté de faire partie de mon jury et d'examiner mon manuscrit.

J'assure de mon entière reconnaissance à ma directrice de thèse Madame Brigitte *Plateau* pour avoir accepté de diriger mes travaux de recherches, pour son accueil et ses qualités humaines. En particulier, je tiens à exprimer ma plus profonde reconnaissance à mes co-directeurs Jean-Marc *Vincent* et Bruno *Dillenseger* pour l'encadrement exemplaire qu'ils ont réalisé. leurs qualités, tant professionnelles qu'humaines, ont permis à notre collaboration de se dérouler de la manière la plus agréable. Chers co-directeurs, Je tiens vraiment à vous remercier pour le temps que vous m'avez consacré, pour votre patience et pour votre soutien aux moments les plus dificiles de cette thèse.

Ensuite, j'adresse des remerciements chaleureux à Nabila *Salmi*, en stage post-doctoral à Orange labs, pour sa collaboration efficace et ses contibutions considérables sur le plan théorique et expérimental de ce travail. Je te remercie Nabila pour ton aide et pour ton optimisme qui m'a beaucoup servi.

Je tiens à remercier toute l'équipe de recherche MAPS/MEP à Orange Labs pour m'avoir fourni des conditions de recherche favorables, tant au niveau humain que matériel. Je remercie également tous les membres des deux équipes du laboratoire LIG (MOAIS et MESCAL), sans citer leurs noms pour ne pas en oublier, pour leur gentillesse et leur accueil.

Je remercie également tous mes collègues à Grenoble de leurs qualités humaines remarquables et pour les moments agréables que nous avons partagé. Je pense particulairement à Mohamed Slim *Bouguerra*, Bilel *Masmoudi*, Rodrigue *Chakode*, Charbel *El Kaed*, Fabrice *Salpetrier* et Ahmed *Ben Farah*.

Je remercie du fond du coeur et avec un grand amour mes chers parents qui n'ont jamais cessé de croire en moi pendant toutes les années d'études et qui m'ont incité avec leurs encouragements et leurs prières d'aller toujours vers l'avant. Un grand merci pour ma soeur, mon frère, à toute la famille et à ma belle famille.

Je ne terminerai pas sans exprimer ma profonde reconnaissance à ma femme que je remercie pour sa patience, son aide considérable et tout l'amour qu'elle me porte. Enfin, un grand merci

à la plus belle des petites filles, mon ange Eya.

Table des matières

Introduction **1**

I État de l'art **7**

1 Concepts d'« Autonomic Computing » et dimensionnement **9**
1.1 Introduction . . . 9
1.2 Autonomic computing . . . 9
1.2.1 Système autonome . . . 10
1.2.2 Architecture d'un système autonome . . . 11
1.2.3 Propriétés autonomes . . . 12
1.3 Dimensionnement . . . 12
1.3.1 Indices de performance . . . 13
1.3.2 Service Level Objective (SLO) . . . 13
1.3.3 Problèmes liés au dimensionnement . . . 14
1.4 Tests de performance . . . 16
1.4.1 Types de Test . . . 16
1.5 Modélisation des systèmes répartis . . . 18
1.5.1 Niveau de détails de modélisation . . . 18
1.5.2 Modèles de performance . . . 19
1.6 Conclusion . . . 19

2 Les réseaux des files d'attentes **21**
2.1 Introduction . . . 21
2.2 Files d'attente . . . 21
2.2.1 Notation de Kendall . . . 22
2.2.2 Caractéristiques d'une file d'attente . . . 23
2.3 Réseaux de files d'attente . . . 25
2.3.1 Classification des réseaux de file d'attente . . . 25
2.3.2 Analyse des performances . . . 26
2.4 Files d'attente simples et principaux résultats . . . 28
2.4.1 M/M/1 . . . 29
2.4.2 M/M/k . . . 29

2.4.3 M/M/1/C 30
2.4.4 M/G/1 30
2.4.5 G/G/1 31
2.4.6 G/G/k 32
2.5 Réseaux des files d'attente et algorithmes de résolution 33
2.5.1 Algorithmes de résolution des réseaux de files d'attente à forme produit 33
2.5.2 Algorithmes de résolution des réseaux de files d'attente à forme non-produit 38
2.6 Simulation des réseaux de files d'attente 42
2.7 Conclusion 44

3 Modélisation et dimensionnement 45
3.1 Introduction 45
3.2 Modélisation 45
3.3 Dimensionnement 49
3.4 Détection des goulets d'étranglement 50
3.4.1 Recheche avec les techniques d'apprentissage 50
3.4.2 Recherche statistique des goulets d'étranglements 51
3.4.3 Recherche avec les mesures d'utilisation 51
3.5 Positionnement par rapport à l'existant 52
3.6 Conclusion 53

II Vers un dimensionnement automatique 55

4 Méthode de dimensionnement 57
4.1 Introduction 57
4.2 Vue d'ensemble 58
4.2.1 Objectifs 59
4.2.2 Principe 59
4.2.3 Etapes de l'approche 60
4.3 Problèmes à résoudre 62
4.4 Décomposition en boîtes noires 62
4.4.1 Principe 62
4.4.2 Types de décomposition 63
4.4.3 Contraintes techniques et contraintes de connaissances 64
4.5 Exemples 65
4.5.1 Rubis 66
4.5.2 Mystore 66
4.5.3 Sample Cluster 66
4.6 Isolation des boîtes noires 67
4.6.1 Motivations 67
4.6.2 Caractérisation des boîtes noires 68

4.7 Conclusion 70

5 Expérimentation automatique pour la modélisation 71
5.1 Introduction 71
5.2 Expérimentation manuelle de test en charge 71
5.2.1 Préparation de l'expérimentation 73
5.2.2 Limites de l'approche manuelle 74
5.3 Expérimentation automatique par test de montée en charge autonome 75
5.3.1 Plates-formes de test en charge 75
5.3.2 Profil de charge automatique 76
5.3.3 Contraintes d'automatisation du profil de charge 78
5.3.4 Paramètrage de l'expérimentation 79
5.3.5 Fonctions 81
5.3.6 Correction de la charge maximale de fonctionnement et identification du nombre de serveurs 88
5.4 Choix des mesures collectées 89
5.5 Conclusion 89

6 Identification d'un modèle et rétroaction 91
6.1 Introduction 91
6.2 Description de l'identification du modèle 91
6.3 Identification d'une loi par analyse statistique 92
6.3.1 Identification par les statistiques descriptives 93
6.3.2 Test d'adéquation ou d'hypothèse 95
6.3.3 Test de χ^2 96
6.3.4 Test de Kolmogorov-Smirnov (KS) 97
6.3.5 Test d'Anderson-Darling (AD) 98
6.3.6 Autres tests 99
6.4 Identification de la loi d'inter-arrivées 99
6.4.1 Capture des requêtes en entrée et échantillonnage 99
6.4.2 Analyse de l'échantillon 99
6.5 Identification de la loi de service 101
6.5.1 Calcul de l'échantillon des temps de service 101
6.5.2 Analyse statistique de l'échantillon des temps de service 102
6.6 Exemple 103
6.6.1 Identification de la loi des inter-arrivées 103
6.6.2 Identification de la loi de service 104
6.6.3 Identification du nombre de serveur 104
6.7 Modèle global de la boîte noire 105
6.8 Checklists d'identification d'une loi 106
6.9 Conclusion 107

7 Construction du modèle global et dimensionnement 109
7.1 Choix du modèle de la boîte à utiliser . . . 109
7.2 Composition des modèles de boîtes noires . . . 110
7.3 Validation du modèle global . . . 111
7.4 Exemple de modélisation et de validation . . . 111
7.4.1 Décomposition . . . 112
7.4.2 Paramètrage . . . 112
7.4.3 Isolation des boîtes noires . . . 112
7.4.4 Modèle de la base de données . . . 113
7.4.5 Modèle du conteneur EJB . . . 115
7.4.6 Modèle du conteneur Web . . . 116
7.4.7 Choix et validation du modèle global . . . 116
7.4.8 Efficacité de la décomposition . . . 117
7.5 Du modèle vers le dimensionnement . . . 118
7.6 Détection et suppression des goulets d'étranglement . . . 119
7.7 Conclusion . . . 122

8 Vers une infrastructure de dimensionnement automatique 125
8.1 Introduction . . . 125
8.2 Besoins techniques et fonctionnels . . . 125
8.2.1 Observations et mesures . . . 126
8.2.2 Contrôle de l'expérimentation autonome . . . 127
8.2.3 Traitement statistique des données . . . 128
8.2.4 Inférence et dimensionnement . . . 128
8.3 Architecture globale de l'environnement de modélisation et de dimensionnement . . . 130
8.3.1 Architecture de l'environnement modélisation automatique . . . 130
8.3.2 Architecture de l'environnement de résolution et de dimensionnement . 133
8.4 Implémentation . . . 135
8.4.1 Expérimentation et modélisation automatique . . . 135
8.4.2 Configuration et dimensionnement . . . 139
8.5 Evaluation de l'architecture . . . 141
8.6 Conclusion . . . 142

Conclusion et perspectives 143

Bibliographie 147

A Spécification du fichier de configuration de la politique d'injection 155
A.1 Variables . . . 155
A.2 Expressions . . . 156
A.3 Instructions . . . 156
A.4 Exemple . . . 157

Table des figures

1 Vue simplifiée de notre approche d'aide au dimensionnement : exemple d'une application multi-tiers 4

1.1 Boucle de contrôle autonomique 10
1.2 Architecture d'un système autonome proposée par IBM 11
1.3 Exemple de déplacement d'un goulet d'étranglement 15
1.4 Modélisation de point du vue système d'un serveur Web 18
1.5 Modélisation de point vue composant d'un serveur Web 19

2.1 Station à capacité finie C 24
2.2 Réseau mono-classe ouvert 26
2.3 Réseau mono-classe fermé 26
2.4 Réseau mixte 26
2.5 Différents algorithmes pour la résolution des réseaux de file d'attente à forme produit 36
2.6 Coût d'exécution et taux d'erreurs relatifs aux différentes approximations de l'algorithme MVA 37
2.7 Etape1 : Exemple de détermination de $\lambda_i(n)$ pour le réseau substitué 39
2.8 Calcul du coefficient de variation par la méthode de décomposition pour les réseaux ouverts 42
2.9 Différents algorithmes pour la résolution des réseaux de file d'attente à forme non produit 43

3.1 Fonctionnement général de la méthode HLM 47
3.2 Architecture pour le contrôle de l'utilisation d'un serveur apache 48
3.3 (a) Architecture du contrôleur de la QoS de [65], (b) Modèle de performance du e-commerce utilisé 48

4.1 Approche globale pour la modélisation automatique et l'aide au dimensionnement 58
4.2 Décomposition suivant l'architecture logicielle d'une architecture client/serveur 63
4.3 Décomposition suivant la localité d'une application répartie 64
4.4 L'application Rubis : version EJB 66
4.5 Architecture de l'application *SampleCluster* 67
4.6 Changement du niveau de charge en entrée d'une boîte suite à une modification d'optimisation 68
4.7 Les bouchons logiciels 69
4.8 Utilisation des sondes pour la caractérisation des boîtes noires 69

5.1 Etape 2 : Expérimentation et identification du modèle de chaque boîte noire . . 72
5.2 Allure de la courbe du temps de réponse moyen en fonction de la charge 73
5.3 Exemple d'expérience avec un niveau de charge fixe 74
5.4 Plaforme de test en charge autonome 75
5.5 Architecture de la plate-forme CLIF 77
5.6 Exemple de profil de charge en palier 78
5.7 Montée en charge de l'application Mystore 80
5.8 Montée en charge de l'application Sample Cluster 80
5.9 Temps de réponse moyen pour la G/G/k (pour k=1 et k=2) 82
5.10 Calcul de la longueur du palier nécéssaire 83
5.11 Calcul du temps de la montée et du temps de stabilisation 83
5.12 Evolution des temps de stabilisation en fonction de la charge injectée 86
5.13 Détermination du nombre de serveurs d'une boîte noire 88

6.1 Rapprochement de l'histogramme par la densité d'une loi de probabilité 93
6.2 Limites de l'identification graphique visuelle 94
6.3 Distance de Kolmogorov-Smirnov 97
6.4 Analyse Graphique de l'échantillon des inter-arrivées : histogramme et graphe de probabilité 103
6.5 Analyse graphique de l'échantillon de service : histogrammes des étapes 8,10 et 13 104

7.1 Exemple d'interconnexion de boîtes noires 111
7.2 Décomposition de l'application *SampleCluster* 112
7.3 Isolation de la boîte 1 : base de données 113
7.4 Isolation de la boîte 2 : conteneur EJB 113
7.5 Isolation de la boîte 3 : conteneur Web 113
7.6 Analyse de l'échantillon de la boîte Base de données : à charge faible(gauche), moyenne (milieu) et forte (droite) 114
7.7 Analyse de l'echantillon de la boîte EJB : à faible (gauche), moyenne (milieu) et forte (droite) charge 116
7.8 Analyse de l'échantillon de la boîte Web : à charge faible (gauche), moyenne (milieu) ou forte(droite) 117
7.9 Temps de résolution et précision du modèle global 118
7.10 Plusieurs configurations envisagées pour éviter un goulet d'étranglement 120

8.1 Vue fonctionnelle de l'environnement d'expérimentation et de modélisation . . 130
8.2 Diagramme de séquence associé au scénario d'exécution 133
8.3 Vue fonctionnelle de l'environnement de configuration et de dimensionnement . 134
8.4 Architecture Fractal de CLIF 136
8.5 Ajout des composants contrôleur et moniteur 136
8.6 Modification du ClifApp.fractal et adaptation à la nouvelle architecture 137
8.7 Définition et description des interfaces du composant "Controller" 137
8.8 Architecture Fractal du composant «Controller» 138
8.9 Fractal ADL du composant «Controller» 138
8.10 Outil de configuration : choix de la configuration à étudier 140

8.11 JModel : réseau de file d'attente associé à la reconfiguration à simuler 140
8.12 JSIM : estimation du temps de service du réseau de file d'attente par simulation 141
8.13 JMVA : calcul de temps de séjour par station par analyse MVA 141

A.1 Exemple de fichier de controle CTL 158

Liste des tableaux

2.1 Coût d'exécution et consommation mémoire 37

5.1 Mystore : temps de stabilisation en fonction de la charge injectée 87
5.2 Rubis : temps de stabilisation en fonction de la charge injectée 87

6.1 Analyse d'un échantillon de temps de service 105
6.2 Résultats des P-valeurs pour chaque palier d'injection 105

7.1 Résultats de l'identification de modèle de la boîte base de données 114
7.2 Résultats de l'identification de modèle de la boîte EJB 115
7.3 Résultats de l'identification de modèle de la boîte Web 116
7.4 Comparaison entre les indices de performance théoriques et empiriques 117
7.5 Comparaison entre les indices de performance théoriques pour les modèles avec ou sans décomposition 117
7.6 Tableau récapitulatif des indices de performance pour différentes configurations 119

8.1 Tableau de comparaison des injecteurs de charge 127
8.2 Tableau de comparaison des solveurs de RFA 129

Introduction

Contexte

Les systèmes répartis, et particulièrement les applications basées sur Internet, ont évolué rapidement ces dernières décennies en intégrant des nouveaux outils et des nouveaux services. Cette évolution engendre naturellement une complexité croissante de ces systèmes. Selon une étude menée par *Network Instruments, près de 70% des administrateurs informatiques s'inquiètent de la complexité croissante des réseaux. Ils sont aussi nombreux à se soucier de l'augmentation du volume de trafic sur le réseau, près de la moitié d'entre eux déclarant que leur problème principal est lié au manque d'informations concernant les problèmes réseau et leurs origines*[42]. Cette complexité et l'augmentation de volume de trafic est également valable pour les applications réparties utilisant ces réseaux.

La non maîtrise du dimensionnement de ces applications réparties peut entraîner de long temps de réponse, une dégradation de la qualité de service ou même une saturation du système causant l'indisponibilité du service. Cette indisponibilité entraîne souvent des grandes pertes économiques allant des clients abandonnant leur achat en ligne, jusqu'à la perte des milliers de transactions boursières sans parler des surcoûts d'exploitation pour faire face à d'éventuelles pannes. A titre d'exemple, en février 2009, le service de messagerie *Gmail* a été indisponible pendant 2 heures et demie, affectant plus de 113 millions d'utilisateurs. En juin 2010, une panne sérieuse [1] a affecté la disponibilité du site *Amazon.com* pendant plusieurs heures. Pire encore, en avril 2011, une panne de plusieurs jours [2] chez *Amazon* a perturbé des services d'hébergement dans le cloud (EC2). Selon *Amazon*, ce dysfonctionnement est dû à une mauvaise configuration d'un centre de données en Virginie Nord (Etats-Unis). Des milliers de sites web clients sont devenus indisponibles pendant cette période (le fameux site *Foursquare* de géolocalisation, le réseau social *Reddit*, le site de questions-réponses *Quora*). Le coût moyen estimé d'une panne sur la base du chiffre d'affaire de 2010 de *Amazon.com* s'élève à 51.400 dollars/minute. La panne de 2010 est alors estimée à 9 millions de Dollars. Aucune estimation n'a été fournie pour la panne de 2011. Toutefois, cette panne reste sans précédent pour *Amazon* vu sa longue durée et le dédommagement des entreprises clientes.

Afin d'éviter ces pertes économiques, les systèmes requièrent de la sûreté et de la fiabilité.

1. news.cnet.com/8301-1023$_3 - 20009241 - 93.html$
2. news.cnet.com/8301-30685$_3 - 20056029 - 264.html$

Ces garanties ne peuvent, en général, être assurées que par des experts qui installent, configurent, optimisent et maintiennent ces systèmes. Cependant, un constat donné dans [28] indique que 40% des pannes de service sont dues à des problèmes de configuration de la part de l'administrateur, et que 80% des budgets sont dépensés dans des opérations de maintenance et dans des améliorations mineures. Malgré leurs coûts, ces améliorations sont généralement basées sur l'intuition et ne permettent ni la localisation du problème de performances ni son contournement. C'est ce que IBM[25] appelle l'*e-panic* quand des administrateurs rajoutent des ressources sans que le problème soit résolu. Cependant, cette difficulté est causée par deux niveaux de complexité :

- **Complexité au niveau de l'architecture et des services :** les infrastructures modernes sont composées de plusieurs systèmes hétérogènes, distribués et orientés service. Au niveau applicatif, des services tels que *youtube, facebook, twitter* ont transformé la page web d'hier formée d'un ensemble de textes et d'images en des plates-formes de partage des vidéos, des réseaux sociaux, de *cloud computing*. Cette complexité induit de fortes contraintes sur les performances vues par l'utilisateur ou par le fournisseur de service.
- **Complexité au niveau des échanges de données :** au vu de l'offre de service, le nombre d'utilisateurs a augmenté d'une manière très importante, induisant des charges difficilement prévisibles et fortement variables.

En réponse à ses clients après la panne de 2011, Amazon Web Services a promis d'« augmenter l'automatisation pour éviter que cette erreur se reproduise »[3]. Cependant, pour réduire les coûts d'administration des applications réparties et maîtriser mieux leur complexité, il est nécessaire d'automatiser la configuration, la surveillance, le déploiement et la protection. De tels systèmes sont dits *autonomiques* et l'approche permettant à ces systèmes de gérer eux-mêmes leur complexité est l'*autonomic computing* [40].

Quant à la variabilité du trafic, il est nécessaire d'élaborer une méthodologie pour *dimensionner* dynamiquement les systèmes afin de garantir la qualité de service souhaitée. Le dimensionnement ou la planification des capacités est la procédure permettant de prévoir la charge de travail en surveillant le système, de prédire les performances afin de garantir la qualité de service [44].

Objectifs et méthodologie

L'objectif de cette thèse est de développer une approche d'assistance au dimensionnement des applications réparties. Cette approche consiste à fournir les outils et les méthodologies basés sur des tests automatisés et l'identification automatisée de modèle de performance.

Etant donné un système informatique complexe, il est conseillé de décomposer le système en plusieurs composants afin de réduire cette complexité. Dans le contexte du dimensionnement des systèmes répartis, cette décomposition devient même fondamentale pour une meilleure lo-

3. http ://aws.amazon.com/message/65648/

calisation des problèmes de performances. Elle doit être faite de manière à séparer en composants élémentaires, étudier leurs performances séparément et décrire les interactions entre les composants. Ensuite, chaque composant est considéré comme un élément atomique (grain minimal de connaissance qu'on peut avoir sur le système) qu'on qualifie de « boîte noire ». La modélisation d'un tel composant ne peut pas se faire en se basant sur des connaissances des mécanismes internes mais uniquement sur des observations d'entrée/sortie (stimulus-réponse). Des tests d'injection de charge sont réalisés au niveau de chaque boîte noire pour en mesurer les performances et les modéliser. Les injections doivent permettre de tester les composants sous différents régimes de charge afin d'avoir un échantillon de mesures complet et représentatif, qui permettra de déterminer les limites opérationnelles de chaque composant. Les phases d'injection de charge et de modélisation doivent être automatisables afin de répondre aux besoins de dimensionnement dynamique et de réduire le coût et la durée des expérimentations.

Une fois tous les composants formant le système modélisés, le modèle global est construit en ajoutant le réseau des interconnexions qui décrivent les échanges inter-composants. Une dernière phase consiste alors à résoudre le modèle global afin de vérifier si la reconfiguration associée au modèle répond au mieux aux besoins de performances.

Contribution de cette thèse

Notre approche consiste à décomposer le système en un ensemble de composants élémentaires. Ensuite, nous automatisons l'expérimentation en utilisant des tests d'injection de charge auto-régulées permettant de produire des mesures représentatives du comportement de chaque composant.

La figure 1 représente une vue sommaire de notre approche. On commence par décomposer une application répartie en plusieurs sous-composants boîtes noires. Ensuite, chaque composant est testé et modélisé automatiquement. Les modèles produits par cette phase permettent à l'expérimentateur de tester plusieurs configurations possibles par combinaison des modèles élémentaires des boîtes noires. Enfin, l'expérimentateur choisit une meilleure configuration qui répond au mieux aux besoins de performances.

Les contributions de cette thèse peuvent être classées comme suit :

Méthodologie de dimensionnement : plusieurs travaux[96, 59, 6, 66] ont fourni des solutions de dimensionnement basées sur des modèles de files d'attente, mais à notre connaissance très peu de travaux se sont intéressés à fonder une approche de dimensionnement complète allant de l'expérimentation et de la collecte de mesures jusqu'à la résolution du modèle et le dimensionnement du système.

Une expérimentation automatique : nous avons mis en place une approche d'expérimentation et un ensemble des mécanismes nécessaires à l'automatisation de l'expérimentation par injection de charge. Cette injection est auto-controlée à travers un modèle de files d'attente permettant de découvrir les limites du système sous test et d'obtenir des mesures fiables et

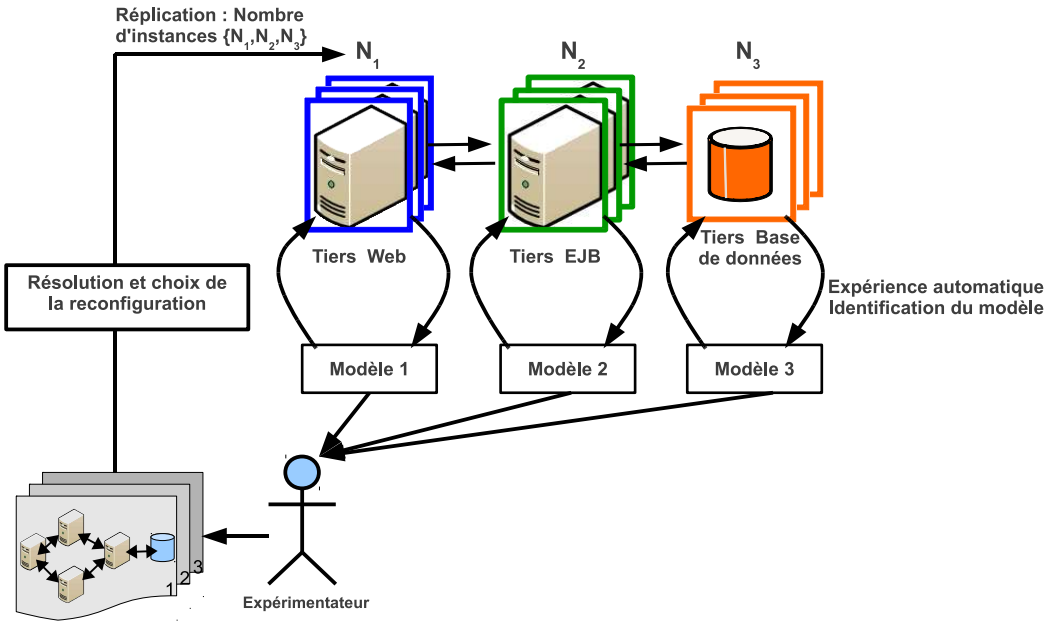

FIGURE 1 – Vue simplifiée de notre approche d'aide au dimensionnement : exemple d'une application multi-tiers

représentatives du comportement global du composant à différents niveaux de charge.

Une approche de modélisation automatique notre approche de modélisation est générique et automatique. Après avoir décomposé un système réparti en un ensemble de composants « boîtes noires », notre approche permet de modéliser automatiquement chaque boîte noire par un modèle de file d'attente à travers un processus d'identification se basant sur les mesures et les tests d'hypothèse statistiques. Le processus d'identification n'émet aucune hypothèse sur le modèle et permet d'identifier des modèles avec un niveau de précision demandé.

Un environnement d'aide au dimensionnement le dernier apport de cette thèse est le développement d'un « framework » générique permettant l'expérimentation et la modélisation automatique des différentes parties constituant un système réparti. Le « framework » permet aussi l'interfaçage avec des outils de résolution et de simulation des réseaux de files d'attente permettant la résolution du modèle du système global et son dimensionnement.

Organisation du manuscrit

Ce manuscrit est composé de deux parties.

Dans la première partie, le chapitre 1 introduit les notions essentielles liées à l'« autonomic computing », le dimensionnement et l'analyse des performances des systèmes répartis. Dans le chapitre 2, on présente le formalisme des réseaux de files d'attente et les principaux résultats sur lesquels se base notre travail. On présente aussi un aperçu des méthodes de résolution ana-

lytique et de simulation des réseaux de files d'attente. Pour terminer cette partie, le chapitre 3 est consacré aux différents travaux existants de modélisation, d'expérimentation automatique, de dimensionnement et d'évitement des goulets d'étranglement. A la fin de ce chapitre, on présente une discussion et un positionnement de notre approche par rapport à l'existant.

La deuxième partie de cette thèse décrit notre approche de dimensionnement en cinq chapitres. Le chapitre 4 présente une vue d'ensemble des différentes étapes de notre approche et explique les problèmes de décomposition et d'isolation qui sont des phases préparatoires de notre approche.

Le chapitre 5 présente le principe d'une expérimentation manuelle, en montrant ses limites, puis détaille notre approche d'automatisation des expérimentations en vue d'une identification de modèle de performance. Deux problèmes fondamentaux sont traités. Le premier est lié à la détermination des mesures qui doivent être à la fois fiables et représentatives du comportement complet (différents niveaux de charge). Le second problème est lié aux mécanismes et aux outils nécessaires à l'automatisation de cette phase expérimentale, afin de réduire la durée des expérimentations sans perdre en qualité d'expérience.

Le chapitre 6 présente la phase de l'identification du modèle de file d'attente d'une boîte noire. Cette identification concerne la loi des inter-arrivées des requêtes, la loi de service et le nombre de serveurs de file d'attente (degré de parallélisme).

Le chapitre 7 présente comment combiner différents modèles de boîtes noires afin de produire un modèle global et d'étudier ses performances. La deuxième partie de ce chapitre est dédiée au dimensionnement par analyse ou par simulation suivant le modèle global obtenu et la qualité du dimensionnement souhaitée.

Le chapitre 8 présente notre *framework* FAMI implémentant notre approche. Après avoir exposé les besoins et les contraintes techniques de l'automatisation et de la modélisation, nous présentons les choix des infrastructures et des outils qui ont permis de répondre à ces besoins de manière générique et extensible. La dernière partie de ce chapitre est consacrée à la présentation de l'architecture de notre framework et son évaluation.

Le manuscrit se termine par une synthèse du travail et une présentation de ses perspectives à court et à long terme.

Première partie

État de l’art

Chapitre 1

Concepts d'« Autonomic Computing » et dimensionnement

1.1 Introduction

Ce chapitre introduit les notions essentielles liées à l'« Autonomic Computing », le dimensionnement et l'analyse des performances d'un système. Dans un premier temps, nous présentons l'autonomic computing, ainsi que les propriétés inhérentes.Dans un deuxième temps, nous donnons les concepts liés aux performances et aux dimensionnements. Nous détaillons ensuite les différents types de test de performance. Enfin, nous terminons par des notions liées à la modélisation des performances.

1.2 Autonomic computing

L'*Autonomic computing* est un concept introduit par IBM en 2001 qui s'est inspiré de la biologie et plus précisément du système nerveux humain. Notre système nerveux gère efficacement notre corps sans aucune intervention de notre part [48]. Il régule le battement de notre cœur, notre température interne et notre débit respiratoire. Malgré leur vitalité et leur récurrence, ces activités sont totalement inconscientes pour l'humain. Ce qui lui permet de consacrer sa pensée à des tâches de plus haut niveau.

Par analogie, on peut définir l'autonomic computing comme suit :

Définition 1.2.1 (Autonomic Computing)**.** *L'*Autonomic computing *est la capacité d'un système à s'auto-gérer, étant donnés des objectifs de haut niveau. Les capacités d'auto-gestion accomplissent leurs fonctions en décidant quelle action appropriée à entreprendre pour une ou plusieurs situations détectées dans leur environnement.*

Suscité par la croissance et la complexité des systèmes informatiques, l'autonomic computing vise à gérer automatiquement les systèmes informatiques, diminuer le coût de leur maintenance et garantir la disponibilité des services et des ressources. Les systèmes dotés de cette capacité sont dits *autonomes*.

1.2.1 Système autonome

L'administration des systèmes actuels, à savoir la configuration, la surveillance, le déploiement et la protection, sont devenus des tâches très difficiles, voire impossibles à assurer manuellement. De plus, plusieurs études montrent que l'être humain est l'une des principales sources d'incidents causés aux systèmes [71]. Ceci a fait naître le besoin de construire des *systèmes autonomes*.

Définition 1.2.2. *Un système autonome est un système capable de s'auto-gérer, d'assurer des tâches d'administration potentiellement complexes et récurrentes et de s'auto-adapter aux changements imposés par son environnement.*

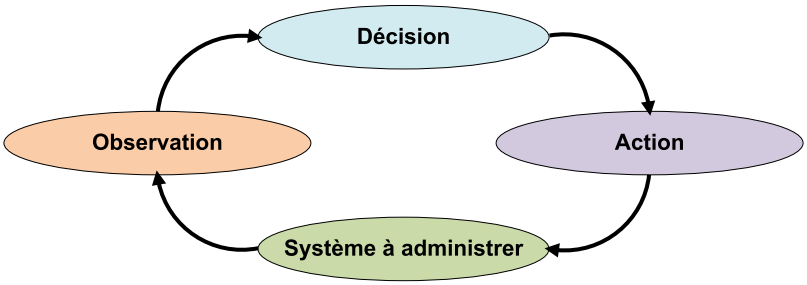

FIGURE 1.1 – Boucle de contrôle autonomique

Pour réaliser des tâches d'une manière autonome, les systèmes autonomes sont basés sur la notion de *boucle de contrôle* (voir figure 1.1). Largement étudiées dans l'automatique, les boucles de contrôle sont basées sur le principe de rétroaction qui se fait en trois phases :

- Une phase d'observation, où un événement ou un problème donné est détecté,
- Une phase de décision suite à l'événement, qui permet de choisir quelle action à entreprendre pour résoudre le problème détecté, et
- Une phase d'action ou de réaction exécutant la décision prise.

Exemple 1.2.1. *Considérons un système distribué constitué d'un ensemble de composants. Un des composants ne répond plus à cause d'une panne ou d'une rafale des requêtes en entrée. Le système s'auto-observe périodiquement et dès qu'il détecte ce problème, décide de remplacer ou de dédoubler le composant. Il déclenche alors une action permettant d'appliquer cette décision.*

Ce principe permet de déléguer une grande partie des tâches d'administration au système lui-même et cache aussi la complexité aux administrateurs. Afin de mettre en œuvre la boucle de contrôle, un système autonome doit être formé de composants clés se chargeant chacun d'une fonction donnée. Ceci est détaillé dans ce qui suit.

1.2.2 Architecture d'un système autonome

Afin de réaliser un système autonome, IBM [40] a proposé une architecture de référence, telle que présentée par la figure 1.2. Cette architecture est assez générale, décrivant les différentes entités qui constituent un système autonome et leurs interactions. Elle est composée de :

1. Un *gestionnaire autonome (autonomic manager)*, implémentant la boucle de contrôle autonome décrite précédemment, et
2. Des *éléments à gérer*, pouvant être des ressources techniques (informatiques, réseaux) ou des applications ou morceaux d'application, des services etc.

Le gestionnaire autonome impose aux éléments à gérer, dans le cycle de la boucle de contrôle autonome, un comportement particulier aux éléments gérés.

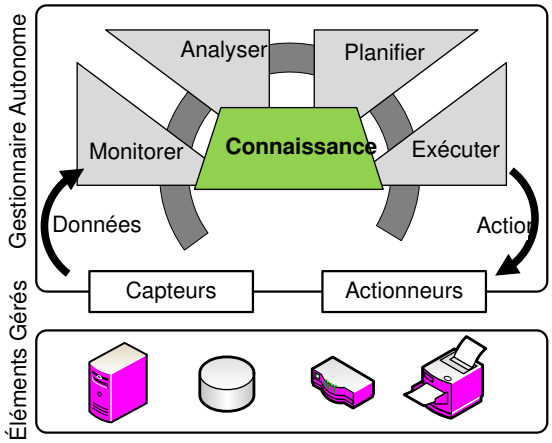

FIGURE 1.2 – Architecture d'un système autonome proposée par IBM

Le gestionnaire autonome est composé de plusieurs parties :

1. *La connaissance :* c'est la partie principale du gestionnaire. Elle permet de sauvegarder les informations relatives aux éléments à gérer à savoir l'historique, le comportement interne et les informations relatives à la politique de gestion de chaque élément à gérer. Cette partie doit mettre à jour régulièrement son contenu.
2. *Le monitoring :* cette partie représente l'interface entre les éléments à gérer et les autres parties du gestionnaire autonome. Elle permet de récupérer les données des différents éléments à gérer, les agréger et fournir ces informations à la partie analyse.
3. *L'analyse :* elle récupère les informations de la partie monitoring et les analyse afin d'identifier les informations relatives à un mauvais fonctionnement ou les informations concernant les portions à optimiser ou à protéger.
4. *La planification :* cette partie décide des actions à entreprendre pour réparer ou optimiser ou protéger les éléments à gérer.
5. *L'exécution :* enfin, ce composant récupére l'ensemble des actions à envoyer aux éléments à gérer et les exécute.

Afin d'interagir avec les éléments à gérer, le gestionnaire autonome est doté d'un ensemble de capteurs et d'actionneurs. Les capteurs produisent des informations des éléments à gérer et les actionneurs sont utilisés pour exécuter une action. Pour des raisons de généralité de l'architecture, IBM n'a pas spécifié plus de détails concernant les interactions entre les différents constituants de cette architecture.

1.2.3 Propriétés autonomes

IBM a défini quatre propriétés à garantir par un système dit autonome. Ces propriétés sont :

1. **L'auto-configuration :** permet à un système de s'adapter automatiquement aux conditions imprévisibles en changeant sa configuration, sans perturber le service. Cela peut se faire par un simple ajout ou suppression d'un constituant du système ou par la modification d'un ou de plusieurs paramètres.
2. **L'auto-réparation :** permet à un système de réparer automatiquement une panne. L'auto-réparation consiste à détecter la panne, la diagnostiquer et par la suite la réparer.
3. **L'auto-optimisation :** permet à un système d'optimiser régulièrement sa configuration afin de garantir un fonctionnement optimal et un respect de ses contraintes. Cette propriété peut être pro-active afin d'améliorer sa configuration par rapport à son état précédent, ou réactive pour répondre aux contraintes de son environnement.
4. **L'auto-protection :** permet à un système de détecter, identifier et se protéger contre les virus, les accès non autorisés, les attaques de dénis de service, etc.

La réalisation de chacune de ces propriétés engendre la proposition d'une nouvelle configuration du système autonome permettant de répondre au problème visé. Toutefois, il peut s'avérer nécessaire de pouvoir répondre à la question : « Est-ce que la nouvelle configuration résout effectivement le problème posé, en préservant la qualité de service ou le niveau de performance exigé(e) ? ». L'analyse de performance d'une configuration peut être une étape fondamentale du processus de décision. Que cela soit pour un scénario d'auto-configuration, d'auto-réparation ou autre, une configuration ne doit être appliquée qu'après une étape de validation.

En particulier, lorsqu'une infrastructure est soumise à un nombre imprévisible d'utilisateurs, il est préconisé de procéder à une analyse permettant une planification des ressources nécessaires pour supporter la charge induite, en d'autres termes procéder à un dimensionnement du système analysé. Pour cette raison, nous nous intéresserons dans la suite au problème de dimensionnement et à l'analyse de performance.

1.3 Dimensionnement

Un des problèmes essentiels que rencontre un administrateur d'un système est de permettre à tous les utilisateurs d'accéder aux services demandés, tout en maintenant la qualité de service

exigée, d'une part, et en minimisant les coûts d'autre part. Ceci revient à prévoir un *dimensionnement* adéquat.

Définition 1.3.1. *Le* dimensionnement *(sizing ou capacity planning) désigne le problème de prévoir la charge d'utilisateurs possible, puis de déterminer quelles sont les ressources nécessaires pour maintenir une performance maximale [66].*

Afin de réaliser un dimensionnement d'un système, il est indispensable de pouvoir analyser ses performances et calculer des *indices de performance*.

1.3.1 Indices de performance

Les indices de performances sont des grandeurs qui décrivent le comportement quantitatif d'un système ou d'un modèle. Ces indices peuvent varier suivant la nature du système étudié. Parmi les indices de performances couramment calculés, on cite :

- *Le temps de réponse :* cette grandeur indique le temps moyen d'exécution d'une requête. C'est le temps qui sépare l'envoi de la requête et la réception du résultat par un client.
- *Le débit :* mesure la quantité du travail fournie par un système pendant un intervalle de temps donné. Qu'il s'agisse d'un réseau ou d'un débit disque, cette grandeur est généralement exprimée en nombre de requêtes ou de quantité d'informations traitée par unité de temps.
- *Le taux d'utilisation* : le taux d'utilisation d'un système est une mesure de son occupation. **Exemple 1.3.1.** *Le taux d'utilisation d'un système est de 20% signifie que le système passe en moyenne 20% de son temps à servir des requêtes et le reste du temps à attendre leurs arrivées. Cette grandeur est un facteur qui influe sur le temps de réponse.*

Plusieurs autres indices peuvent être calculés suivant la nature du système et les objectifs de l'étude de performance à réaliser. Par exemple, le calcul des probabilités de rejet et le calcul du nombre moyen des requêtes ou des clients dans un système peuvent être considérés comme des facteurs importants suivant les objectifs visés. Dans d'autres cas, la qualification des performances d'un système avec des indices moyens de performance s'avère insuffisante et peut conduire à des surprises regrettables. Dans ces cas, les valeurs limites, les valeurs maximales et les écarts ou des variances sont indispensables pour une bonne qualification du système.

Dans un contexte industriel, les indices de performance permettent de quantifier la *qualité de service (QoS)* d'un système. Celle-ci définit des *Objectifs de Niveau Service (Service Level Objectives)*.

1.3.2 Service Level Objective (SLO)

La qualité de service offerte par un fournisseur à ses clients est dictée par un *contrat de niveau de service (Service Level Agreement ou SLA)*. Ce contrat spécifie un ensemble de condi-

tions à respecter appelées *Objectif de Niveau de Service*.

Définition 1.3.2. *Un* Objectif de Niveau de Service *ou* Service Level Objective (SLO) *représente une ou plusieurs mesures de la qualité de service d'un fournisseur. C'est la partie technique spécifiant la qualité de service d'un contrat SLA.*

Un SLO peut exprimer la disponibilité d'un service, un temps de réponse ou un débit...

Exemple 1.3.2. *Pour une application de vente en ligne, un SLO peut être : « le temps de réponse ne doit pas excéder en moyenne 4 secondes :* $\overline{R} < 4$ *sec ».*

Un SLO doit être significatif et acceptable par les deux parties du contrat (client et fournisseur) [92]. Il peut être violé lorsqu'une charge importante de requêtes utilisateurs est soumise au système étudié. Ceci constitue un des soucis majeurs du dimensionnement.

Nous allons donner dans la suite les phénomènes liés au dimensionnement pouvant influer sur les SLOs.

1.3.3 Problèmes liés au dimensionnement

Différents phénomènes sont recherchés pour réaliser un dimensionnement adéquat : les goulets d'étranglement, le point de saturation et le point optimal de fonctionnement.

a Goulets d'étranglement

Dans la majorité des systèmes, les performances globales sont les performances imposées par la partie la plus faible du système. Cette partie est appelée goulet d'étranglement [56]. On peut définir alors un goulet d'étranglement comme étant la partie ou le composant du système limitant ses performances globales. La détection des goulets d'étranglement représente un défi important pour toute étude de performance. En effet, une ressource peut représenter un goulet d'étranglement dans certains cas et pas dans d'autres. De plus, un goulet d'étranglement peut en cacher un autre. Les goulets dépendent de tous les paramètres de performance du système et de la charge de travail qui lui est imposée. Ils affectent souvent les temps de réponse du système ou le débit. Dans le cas d'un système multi tiers, le goulet d'étranglement peut être dû au tier base de données ou au tier du serveur d'application suivant le type de l'application déployée et suivant la nature de la charge imposée.

On peut classer les goulets d'étranglement suivant deux catégories :
- Un goulet logiciel est souvent dû à un manque d'optimisation du code, au mécanisme de fonctionnement interne ou à une mauvaise gestion des ressources qui fait réduire considérablement les performances du système.
- Un goulet matériel est souvent dû à un sous dimensionnement d'une partie ou de la globalité de l'architecture physique, qui a un effet sur le fonctionnement global du système.

L'identification des goulets d'étranglement représente la première étape de tout projet d'optimisation des performances.

b Déplacement des goulets d'étranglement

Le problème de déplacement des goulets d'étranglement est un problème fréquent lorsqu'on cherche à dimensionner dynamiquement une infrastructure répartie. Pour expliquer ce problème prennons l'exemple de la figure 1.3.a. Cet exemple illustre une architecture simple d'un systeme 3-tiers et met en evidence le déplacement du goulet d'étranglement. Le goulet d'étranglement pour cet exemple est présent au niveau du serveur d'application qui limite le trafic à 150 requêtes par seconde pour une demande de 250 requêtes par seconde. Une solution intuitive qui repond à ce problème consiste à dupliquer le serveur d'application tel que decrit dans la figure 1.3.b. Le goulet se déplace au niveau du serveur HTTP et limite le trafic à nouveau à 200 requêtes par seconde.

Cet exemple montre que :
- la construction d'une solution à un problème de goulet d'étranglement ne peut pas se faire en dupliquant naïvement le serveur causant le problème.
- la solution doit se faire d'une façon itérative afin de s'assurer que le goulet n'est pas déplacé à un autre niveau.

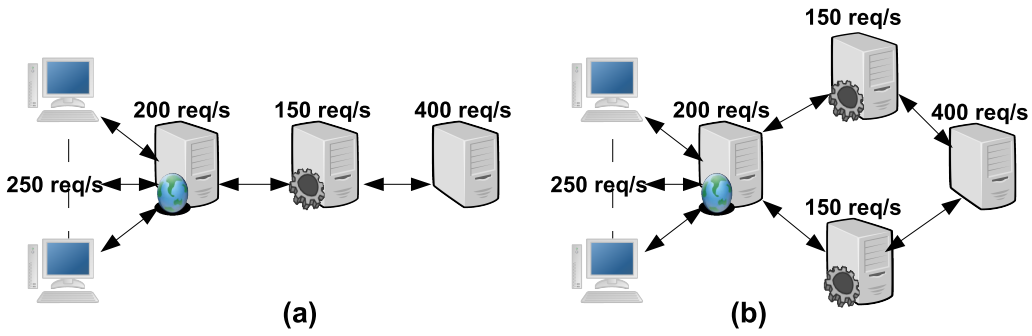

FIGURE 1.3 – Exemple de déplacement d'un goulet d'étranglement

c Point de saturation

Un point de saturation est un goulet d'étranglement lorsque le système est correctement configuré. Les performances du système avant un point de saturation présente les capacités maximales qu'un système peut garantir. Une des caractéristiques d'un système saturé est qu'il est souvent incapable de revenir vers un état stable. Pour cette raison, il est généralement important de prévoir les ressources nécessaires pour ne pas atteindre un état de saturation.

d Point optimal de fonctionnement

Les administrateurs cherchent constamment à assurer un meilleur rendement de leurs systèmes, en évitant les goulets d'étranglement et en ayant une configuration optimale. Pour cela, il est nécessaire de connaître ce qu'on appelle le *Point optimal de fonctionnement*.

Le *point optimal de fonctionnement* est défini comme étant la charge maximale que le système peut supporter en respectant les SLO définis par l'administrateur.

Afin de détecter les problèmes liés aux performances de manière générale, et au dimensionnement en particulier, nous présentons les tests de performance.

1.4 Tests de performance

Un test de performance est un test dont l'objectif est de déterminer la performance d'un système informatique. Il permet d'observer et d'évaluer les réponses d'un système soumis à toutes les conditions de charge possibles. Il existe une variété de tests de performance, définis suivant les spécifications qu'on veut valider.

1.4.1 Types de Test

Les tests de performance ont été classifiés suivant leurs objectifs et les méthodes utilisées. Nous allons nous limiter dans ce chapitre aux définitions des tests de performance les plus connus.

a Classification par méthodes utilisées

a.1 Test de performance Ce test consiste à émuler une charge particulière et de mesurer les performances du système. Plusieurs indices de performance peuvent faire l'objet d'un test de performance. Citons par exemple : le temps de réponse, le débit...

A l'inverse des autres tests, l'objectif ici n'est pas de valider les performances mesurées par rapport à une charge en entrée, mais plutôt de mesurer les performances par rapport à plusieurs niveaux de charge [98].

a.2 Test de tenue en charge L'objectif de ce type de test est de déterminer la capacité maximale du système étudié pour des fins de prévision des performances. Il consiste alors à émuler le comportement de plusieurs utilisateurs (la charge) en assurant une montée en charge

progressive : On part d'un nombre minimal d'utilisateurs, jusqu'à atteindre la charge maximale supportée.

a.3 Test de stress Ce test consiste à générer une activité maximale pour une période plus au moins longue afin de déterminer quand le système sature et d'analyser les conséquences d'une telle défaillance. Les pics de trafics peuvent varier d'un système à un autre ou d'une période à une autre pour un système donné[98].

Par exemple, pour des systèmes de bourse en ligne, la charge maximale est obtenue juste après l'ouverture des marchés ou juste avant leur fermeture. Tandis que cette charge est plutôt observée en fin de journée pour les sites de vente en ligne. Cette observation ne sera plus valable en période de fin d'année. La prévision des pics de trafic est d'autant plus importante lorsque le système étudié utilise des répartiteurs de charge ou des mécanismes de contrôle d'accès.

b Classification par accessibilité

L'accessibilité au code et le niveau de complexité des mécanismes internes d'une application à évaluer sont des éléments clés pour le choix du type de test à utiliser. Les tests de performance peuvent être classés comme suit :

b.1 Test de boîtes noires Une boîte noire est une vue d'un système sans considérer son fonctionnement interne. Le comportement de la boîte noire n'est donc capturé qu'à travers ces interactions. Les tests de performance de type boîte noire sont utilisés pour estimer et vérifier les sorties en fonction des entrées. Ce genre de test est souvent utilisé lorsqu'il s'agit des systèmes dont le comportement interne est difficile à connaître.

b.2 Test de boîtes blanches Contrairement au test de boîtes noires, le test de boîtes blanches se base principalement sur le comportement interne du système. La connaissance du fonctionnement de l'ensemble des éléments qui composent le système permet d'analyser le fonctionnement interne et d'évaluer ses performances. Ce test utilise souvent le code source et les spécifications du système à tester. Surtout lorsque ces spécifications sont disponibles.

b.3 Test de boîtes grises Le test de boîtes grises est un test hybride qui réutilise les deux approches précédemment décrites pour évaluer les performances d'un système sous test. Des scénario de fonctionnement sont généralement établis ou prédits en se basant sur les informations et les spécifications disponibles. Ces tests sont généralement adaptés aux systèmes complexes dont on connaît une partie des comportements internes.

Les tests de performance permettent de détecter certains problèmes. Toutefois, ces tests ne sont pas suffisants, et l'administrateur a besoin de prédire les performances de différentes

configurations de son système afin d'en déduire un dimensionnement adéquat.

La prédiction de performances d'un système s'appuie sur sa modélisation formelle.

1.5 Modélisation des systèmes répartis

Plusieurs travaux ont adressé la modélisation des systèmes répartis. La construction d'un modèle de performance d'un système dépend de deux critères :

1. Le niveau de détails de la modélisation, et
2. Le modèle de performance choisi.

1.5.1 Niveau de détails de modélisation

Un modèle de performance peut être construit selon différents niveaux de détails [66]. Le choix du niveau de la modélisation dépend de plusieurs facteurs :

– La précision des résultats souhaités ;
– Les informations disponibles sur le système ;
– La solvabilité du modèle.

On distingue deux types de modélisation : une modélisation du point de vue composant et une modélisation de point de vue système.

a Modélisation du point de vue système

Dans ce type de modélisation, les détails du système ne sont pas modélisés explicitement. Ainsi, un serveur de base de données complet peut être vu comme une boîte noire et modélisé par une seule entité. Cette modélisation est réalisée lorsque les systèmes à analyser sont très grands.

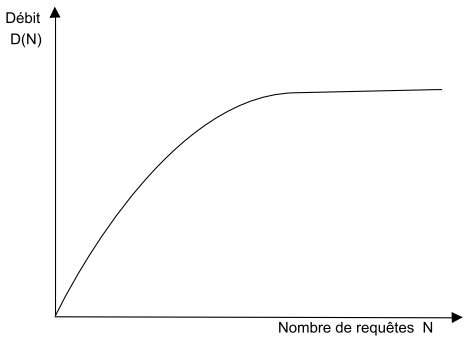

FIGURE 1.4 – Modélisation de point du vue système d'un serveur Web

La figure 1.4 décrit la modélisation d'un serveur web en le considérant comme une boîte noire. Le comportement interne du serveur n'est pas modélisé. Seul le débit en fonction du nombre de requêtes est considéré.

b Modélisation du point de vue composant

Contrairement à la modélisation du point de vue système, la modélisation du point de vue composant prend en considération des informations sur les mécanismes internes du système (architecture, interactions...). On doit alors modéliser les ressources utilisées et leur façon de servir les requêtes. Cette modélisation requiert la connaissance précise du fonctionnement interne du système à modéliser. En utilisant le réseau des files d'attente, un serveur web peut être modélisé de point de vue composant par la figure 1.5.

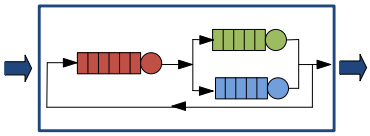

FIGURE 1.5 – Modélisation de point vue composant d'un serveur Web

1.5.2 Modèles de performance

Plusieurs modèles de performance ont été utilisés à des fins de modélisation et d'évaluation des performances d'un système réparti. Le modèle fondamental de l'analyse des performances est le modèle de *réseau de files d'attente* [50]. Avec ce modèle, et pour des cas « relativement simples », il est possible de calculer une solution analytique des probabilités à l'équilibre, puis de calculer des indices de performance en utilisant les probabilités à l'équilibre. En plus des files d'attente, de nombreux modèles plus ou moins spécialisés ont été développés pour l'étude des performances des systèmes informatiques : réseaux de Petri, réseaux d'automates stochastiques, algèbres de processus stochastiques, etc. Le développement de ces modèles avait pour objectif de répondre à la diversité et à la complexité croissante des systèmes étudiés.

1.6 Conclusion

Ce chapitre a présenté les différents concepts de dimensionnement et d'évaluation des performances des systèmes, dans un but autonomique. Le dimensionnement et l'évaluation des performances des systèmes répartis sont des tâches de plus en plus indispensables et critiques. Il n'est plus acceptable que ces tâches se fassent sans une méthodologie bien définie et bien étudiée.

La modélisation des systèmes est une étape fondamentale pour pouvoir prédire et évaluer les performances des systèmes. Pour répondre aux besoinx de modélisation des performances, la communauté scientifique a développé des nombreux modèles. La majorité des travaux de dimensionnement se base essentiellement sur le modèle des réseaux des files d'attente. Ceci est expliqué par plusieurs points forts de ce modèle à savoir :

1. La simplicité : Le modèle de file d'attente est caractérisé par la simplicité de sa représentation (voir la section 2.2).
2. Le partage des ressources : L'adéquation de la représentation de ce modèle à la notion de partage de ressources qui représente un défi majeur de l'évaluation des performances des systèmes répartis.
3. L'évaluation des performances : De nombreux résultats analytiques exacts et approchés existent et permettent de résoudre les modèles des files d'attente analytiquement.
4. L'adaptabilité à des systèmes de type boîte noire : Les réseaux de files d'attente sont parfaitement adéquats pour des systèmes dont on ne connaît pas les détails de fonctionnement interne.

Pour ces raisons et bien d'autres qui seront développées dans les sections 3.2 et 3.3, profondément liés aux types de système qu'on souhaite dimensionner, le modèle de réseau de files d'attente est préconisé. Afin de bien montrer les avantages de ce choix, nous présentons les détails de ce modèle dans le prochain chapitre.

Chapitre 2

Les réseaux des files d'attentes

2.1 Introduction

La modélisation d'un système est l'« Opération par laquelle on établit un modèle d'un phénomène, afin d'en proposer une représentation, interprétable, reproductible et simulable. » [2]. Les *réseaux des file d'attente* [50, 49, 38, 19, 12] constituent un formalisme de modélisation, largement utilisé pour l'évaluation des performances des systèmes à événements discrets tels que les systèmes informatiques, les réseaux de communication et les systèmes de production. Ce modèle permet de représenter la notion de partage des ressources, où une ressource est partagée entre plusieurs clients. Dans notre contexte, la partage des ressources représente la source principale de la majorité des problèmes liés au dimensionnement. Comme le formalisme des files d'attente permet de modéliser convenablement ce partage, la résolution du modèle généré va permettre, à priori, de répondre à de nombreuses questions liées au dimensionnement.

Dans ce chapitre, nous allons présenter le formalisme des réseaux de files d'attente. Nous commençons dans une première section par introduire la notation de Kendall et la formule de Little. Dans une deuxième section, nous donnons les principaux résultats des files d'attente sur lesquels se basent notre travail. Une troisième section sera dédiée aux réseaux de files d'attente, à savoir leurs types et leurs méthodes de résolution analytique. La dernière section présentera la simulation des réseaux de files d'attente, qui constitue un dernier recours lorsque le réseau à analyser ne peut pas être résolu analytiquement.

2.2 Files d'attente

Un *système d'attente simple* ou *station* consiste en une file d'attente, appelée aussi *buffer* ou *tampon*, et d'une station de service constituée d'une ou plusieurs ressources appelées *serveurs*. Des entités dites *clients* ou *travaux (jobs)*, générées par un processus d'arrivée externe,

rejoignent la file pour recevoir un service offert par l'un des serveurs. Une politique de service (généralement FIFS, premier servi de la file) est adoptée pour servir les clients. A la fin du service, le client quitte le système. Un tel système est caractérisé par :

- Le nombre de serveurs,
- la discipline de service,
- la capacité du buffer,
- le processus d'arrivée, et
- le processus de service.

Ces caractéristiques sont définies à travers la notation suivante, dite de *Kendall* [12, 50].

2.2.1 Notation de Kendall

Dans la théorie des files d'attente, la notation de Kendall est un standard utilisé pour décrire un modèle de file d'attente. Une file d'attente s'écrit sous la forme $T/X/K/C/m/Z$ avec :

T : La distribution d'inter-arrivée des clients,

X : La distribution de service,

K : Le nombre de serveurs,

C : La capacité de la station,

m : Le nombre maximum de clients susceptibles d'arriver dans la file d'attente, et

Z : La discipline de service (FIFS, LIFS, PS).

T et X peuvent être données par plusieurs types de distributions. En voici les plus répandues :

M **:** Loi sans mémoire.

E_k **:** Loi de Erlang à k étages.

D **:** Loi constante (déterministe).

H_k **:** Loi hyperexponentielle-k.

C_k **:** Loi de Cox-k.

PH_k **:** Loi de type « phase » à k étages.

G **:** Loi générale.

GI **:** Lois générales indépendantes.

Les valeurs par défaut des trois derniers paramètres de la notation de Kendall sont C=$+\infty$, m=$+\infty$ et Z=FIFS. Dans la suite, nous utilisons la notation T/X/k. Nous expliquons maintenant chacune des caractéristiques d'une file d'attente.

2.2.2 Caractéristiques d'une file d'attente

a Processus d'arrivée

L'arrivée des clients, décrite par le symbole A, est définie à l'aide d'un processus stochastique de comptage N_t avec $t \geq 0$.

Définition 2.2.1. *Soit A_n la date d'arrivée du n^{ime} client dans le système :*

$$A_0 \equiv 0 \text{ et } A_n \equiv inf\{t|N_t = n\}.$$

Soit T_n le temps séparant l'arrivée du $(n-1)^{ime}$ client et celle du n^{ime} client. La loi de T_n est dit distribution des inter-arrivées *:*

$$T_n = A_n - A_{n-1}$$

Définition 2.2.2. *Un processus de comptage N_t avec $t \geq 0$ est un processus de renouvellement si et seulement si les variables aléatoires $(T_n)_{n=1,2,...}$ sont des variables indépendantes et identiquement distribuées.*

La loi de T décrivant les durées d'inter-arrivées suffit alors pour caractériser le processus de renouvellement.

Remarques 2.2.1. *Notons que, lorsque les inter-arrivées sont de loi exponentielle, le processus d'arrivé est un processus de Poisson. Ce dernier est le processus le plus couramment employé pour caractériser les processus d'arrivée.*

b Temps de service

Le temps de service est défini par le temps séparant le début de la fin du service. On note X_n le temps de service du n^{ime} client.

Remarques 2.2.2. *La distribution du temps de service la plus couramment utilisée est la distribution exponentielle, qui est caractérisée par la propriété sans mémoire.*

c Le nombre des serveurs

Une station peut contenir un ou une infinité de serveurs. Dans le cas multiserveur, la détermination de la distribution de service de chacun des serveurs est recommandée voire indispensable. La plupart du temps, les serveurs sont considérés identiques et indépendants les uns des autres.

Pour modéliser un phénomène de retard pur ou un système dont le nombre de serveur est toujours supérieur au nombre de client, on utilise une station avec un nombre infini de serveur. L'attente est alors réduite à zéro et temps de séjour dans la station est égal au temps de service.

d La capacité de la station

La capacité d'une station notée C est le nombre de clients maximal qui peut se trouver simultanément dans la file et les serveurs. Cette capacité peut être finie ou infinie. Par conséquent, si un client arrive et qu'il y a déjà C clients dans le système, le client peut être accepté ou rejeté suivant la politique de débordement de la station (voir figure 2.1).

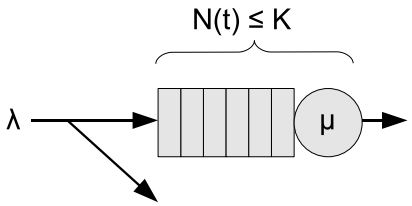

FIGURE 2.1 – Station à capacité finie C

e La discipline de service

L'objectif de la discipline de service est de déterminer l'ordre de service des clients dans la file d'attente et leurs passages dans les serveurs, afin d'être servis. Les disciplines les plus courantes :

- FIFO(First In First Served) : Les clients sont servis dans leur ordre d'arrivée,
- LIFO(Last In First Served) : Le dernier client arrivé sera placé en tête de file pour être servi en premier,
- PS (Processor Sharing) : Cette discipline est définie pour modéliser des systèmes informatiques. Tous les clients sont servis à tour de rôle. Chaque client effectue un quantum de temps très petit dans le serveur et revient dans la file d'attente jusqu'à terminer complètement son service.
- Random : un ordre aléatoire d'accès aux serveurs,
- etc.

Dans cette première partie nous avons introduit la notation et les caractéristiques d'une file d'attente simple qui nous permettra de modéliser les systèmes. Toutefois, la majorité des systèmes étudiés sont assez complexes pour qu'ils soient modélisés par une seule file d'attente simple, d'où la notion de *réseau de files d'attente*.

2.3 Réseaux de files d'attente

Un réseau de files d'attente est un ensemble de file d'attentes interconnectées. La définition de ce réseau requiert, d'une part, la définition de toutes les files qui le constituent, et d'autre part, la définition du routage entre ses files. En effet, après avoir terminé son service dans un serveur, un client entre dans une autre station ou quitte définitivement le réseau. Plusieurs types de routage existent :

- Routage probabiliste : un client sortant d'une station i a une probabilité p_{ij} à passer à la station j ou une probabilité p_{i0} pour quitter le réseau. L'ensemble des probabilités de routage de toutes les stations sont regroupées dans une matrice de routage.
- Routage dynamique : est un routage qui se fait suivant l'état de système au moment du routage. Par exemple, un client sortant d'une station i peut choisir la station comportant moins de clients parmi ces destinations possibles.
- Routage cyclique : un client sortant d'une station choisira à tour de rôle une de ces destinations possibles.

Un réseau de files d'attente peut être classé selon les types ou le nombre de classes de clients qui le parcourent ou suivant sa topologie.

2.3.1 Classification des réseaux de file d'attente

Un réseau de file d'attente est un ensemble de files d'attente interconnectées, dans lesquelles circulent une ou plusieurs classes de clients. Chaque classe se caractérise par un schéma de routage, par des comportements différents au niveau de chaque station, de service et de l'ordonnancement dans le buffer d'attente. On peut distinguer différentes classes de clients suivants [12] :

- des processus d'arrivées différents ;
- des comportements des clients qui sont différents à chaque station ;
- des routages différents dans le réseau ;

Un réseau parcouru par une seule classe de clients est appelé réseau *mono-classe*. Contrairement au réseau *multi-classe* dans lequel circulent plusieurs classes. Chaque classe de clients peut être soit ouverte soit fermée.

a Réseau mono-classe ouvert

On appelle réseau mono-classe *ouvert* un réseau avec une seule classe de client où les clients arrivent de l'extérieur du système, séjournent pour recevoir un ou plusieurs services, puis quittent définitivement le système. Par conséquent, le nombre de clients est infini. La figure 2.2 montre un exemple de réseau mono-classe ouvert.

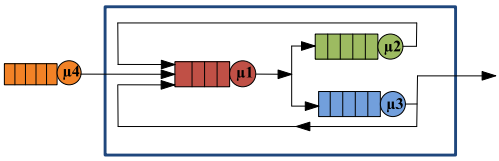

FIGURE 2.2 – Réseau mono-classe ouvert

b Réseau mono-classe fermé

Un réseau est dit mono-classe *fermé* lorsque tous les clients appartiennent à la même classe et leur nombre en entrée du système est constant. Il n'y a ni de départ ni d'arrivée de clients. La figure 2.3 montre un exemple de réseau mono-classe fermé.

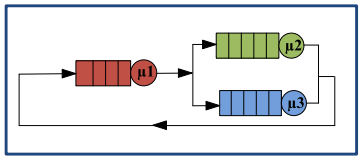

FIGURE 2.3 – Réseau mono-classe fermé

c Réseau mutli-classe mixte

On appelle réseau multi-classe *mixte* un réseau contenant au minimum une classe de clients ouverte et une classe fermé. La population de clients externes est infinie, alors que la population de clients internes est finie. La figure 2.4 montre un exemple de réseau multi-classe mixte.

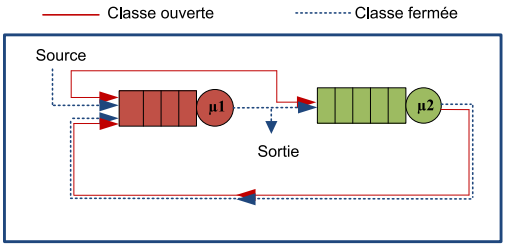

FIGURE 2.4 – Réseau mixte

Remarques 2.3.1. *Un réseau multi-class contenant que des classes ouvertes (fermées) est appelé* purement ouvert*(*purement fermé*).*

2.3.2 Analyse des performances

Les réseaux des files d'attente servent à analyser les performances du système modélisé. Cette analyse peut être menée en étudiant le comportement du système selon deux axes :

1. *Étude en régime transitoire :* L'étude du régime transitoire permet de répondre à des questions de performance qui sont liées à des instants donnés ou sur des périodes de court terme. Par exemple, « combien de clients demandant un service X vont être servis durant la prochaine heure ? ».

2. *Étude en régime stationnaire ou permanent (dit aussi à l'équilibre) :* consiste à vérifier si le système tend vers un équilibre (en terme de probabilité) lorsque le temps croît (à long terme). Cette analyse permet de répondre aux questions telles que : durant une longue période, quel est le taux moyen d'occupation du serveur ?

Pour étudier cela, des méthodes stochastiques sont utilisées. Elles consistent à estimer la distribution du processus stochastique engendré par le modèle analysé, soit à un instant donné (analyse transitoire), ou bien à long terme (à l'équilibre). Elles permettent de calculer les probabilités pour que le système se trouve dans chacun des états du processus. Ces probabilités sont utilisées pour le calcul des paramètres de performance.

Les modèles de files d'attente les plus simples à analyser sont les modèles *markoviens* (distributions exponentielles des inter-arrivées et service). Ceux-ci engendre une chaîne de Markov (d'où le nom markovien).

a Paramètres de performance en régime stationnaire

Différents paramètres de performance peuvent être défini puis calculé en régime transitoire ou en régime permanent. En régime transitoire, les moyennes de ces paramètres sont calculés sur une période d'observation finie $[0, T_{obs}]$. Alors que dans le régime permanent, on s'intéresse aux limites de ces paramètres lorsque T_{obs} tend vers l'infini, si cette dernière existe.

- *D : Le débit moyen* c'est le nombre moyen des clients entrants ou sortants du système par unité de temps. Dans le régime transitoire, le débit moyen d'entrée peut être différent du débit moyen de sortie. Contrairement au régime permanent où le débit moyen de clients est le même en entrée qu'en sortie.
- *U : L'utilisation moyenne* où le taux d'utilisation moyenne est défini comme le pourcentage pour lequel le système (le serveur) est occupé par rapport au temps d'observation.
- *N : Le nombre de clients moyen* est le nombre moyen des client présents dans le système pendant la période d'observation.
- *R : Le temps de réponse moyen* est le temps de séjour moyen d'un client dans le système.

Pour un réseau de files d'attente, ces paramètres peuvent être calculés en considérant le réseau entier. Par exemple le temps de réponse moyen d'un client sera le temps moyen nécessaire pour qu'un client traverse tout le réseau. Dans le cas d'un réseau multi-classe, ce paramètre doit être calculé pour chaque classe de client.

b Stabilité

La stabilité d'un système est définie qu'au régime stationnaire. On suppose qu'il n'y a ni création ni destruction des clients en interne. Cela implique que toute création de client correspond à une arrivée et toute destruction correspond à un départ.

b.1 Cas d'une file simple

Définition 2.3.1. *Une file d'attente simple ayant un taux d'arrivé λ, un taux de service μ et un nombre de serveur k est dite stable si et seulement si $\lambda \leq k \times \mu$.*

b.2 Cas d'un réseau de files d'attente ouvert Un réseau de file d'attente ouvert est stable si et seulement si toutes les stations qui le constituent sont stables [12].

c Formule de Little

La formule de Little [60] est une loi générale qui s'énonce comme suit : « le nombre moyen des clients dans un système N est égal au produit du débit du système D par le temps moyen passé dans le système par chaque client R » [51].

Un client arrivant trouve en moyenne N clients devant lui. Ce client partant laisse derrière lui $R * D$ clients. Donc dans l'état stationnaire :

N=D*R

. L'importance de la formule de Little réside dans sa généralité [60]. En effet, elle peut s'appliquer sur :

- les files d'attente incluant buffers et serveurs,
- le buffer de la file seulement,
- le(s) serveur(s) de la file seulement.

De plus, cette loi ne concerne que le régime permanent. Elle n'impose aucune hypothèse ni sur le système, ni sur les variables aléatoires qui le caractérisent.

Dans la suite, nous intéressons aux principaux résultats de performances pour les files simples, puis pour les réseaux de files d'attente.

2.4 Files d'attente simples et principaux résultats

Nous présentons dans cette section les principaux résultats des files simples en partant du modèle markovien jusqu'au modèle le plus général (où les inter-arrivées et les services sont

arbitraires avec plusieurs serveurs). Nous allons nous limiter aux files et aux résultats dont on aura besoin dans la suite de ce manuscrit.

2.4.1 M/M/1

La file M/M/1 est la file la plus simple et la plus utilisée pour modéliser les systèmes informatiques. L'utilisation de cette file est motivée par l'ensemble de ses résultats permettant de déterminer les paramètres de performances moyens. Elle est définie par le processus stochastique suivant :

- le processus d'arrivée des clients est distribué selon un processus de Poisson de paramètre λ.
- le processus de temps de service est indépendant du processus d'arrivée et suit la loi exponentielle de paramètre μ.
- un seul serveur.

On peut calculer les probabilités du régime transitoire et celles du régime permanent, ainsi que tous les paramètres de performance en utilisant les chaînes de Markov[12, 50, 18]. Soit $P(0)$ la probabilité pour que le système ne contient aucun client en régime permanent et on note $\rho = \frac{\lambda}{\mu}$. Les paramètres de performance moyen de cette file en régime stationnaire sont :

- Le débit moyen D : $\mathbf{D} = [1 - P(0)]\mu = \rho\mu = \lambda$
- Le taux d'utilisation U : $\mathbf{U} = [1 - P(0)] = \rho$
- Le nombre moyen des clients N :

$$\mathbf{N} = \frac{\rho}{(1-\rho)}$$

- Le temps moyen de séjour R :

$$\mathbf{R} = \frac{N}{D} = \frac{1}{\mu(1-\rho)} = \frac{1}{\mu} + \frac{\rho}{\mu(1-\rho)}$$

2.4.2 M/M/k

Pour le cas markovien avec plusieurs serveurs, les paramètres de performance moyens en regime stationnaire sont :

- Le débit moyen D : $\mathbf{D} = \lambda$
- Le nombre moyen des clients N : $\mathbf{N} = RD = R\lambda$
- Le temps moyen de séjour R :

$$\mathbf{R} = P(0)\frac{\rho^k}{\mu(k-\rho)^2(k-1)!} + \frac{1}{\mu}$$

avec μ est le taux de service d'un serveur et tous les serveurs sont identiques.

2.4.3 M/M/1/C

Une file M/M/1/C est identique à une file M/M/1, à l'excéption de la capacité de la file considérée dans ce cas comme finie.

Les paramètres de performances spécifiques à cette file sont comme-suit :
- Le débit moyen : $\mathbf{D} = \frac{1-\rho^C}{1-\rho^{C+1}}\lambda$
- Le taux d'utilisation : $\mathbf{U} = [1 - P(0)] = \rho\frac{1-\rho^C}{1-\rho^{C+1}}$
- Le nombre moyen des clients : $\mathbf{N} = \sum_{n=0}^{C} np(n)$

$$\mathbf{N} = \frac{\rho}{1-\rho}\frac{1-(C+1)\rho^C + C\rho^{C+1}}{1-\rho^{C+1}}$$

- Le temps moyen de séjour des clients non rejetés :

$$\mathbf{R} = \frac{\mathbf{N}}{\mathbf{D}} = \frac{1-(C+1)\rho^C + C\rho^{C+1}}{(\mu-\lambda)(1-\rho^C)}$$

2.4.4 M/G/1

La file M/G/1 est une file à capacité illimitée ayant un seul serveur. Son processus d'arrivée suit la loi poissonienne de taux λ tandis que le temps de service est distribué selon une variable aléatoire **générale** G. Dans ce cas particulier, on peut déterminer tous les paramètres de performances moyens, même si son service ne vérifie pas la propriété sans mémoire. Ces paramètres sont déterminés grâce à la méthode de la *chaîne de Markov induite*. Cette méthode consiste à ramener l'étude à une chaîne de Markov à temps discret en considérant des instants d'observation particuliers (instants de début de service ou instants de fin de service). Les paramètres de performance moyens se présentent alors comme suit :

- Le débit Moyen : $\mathbf{D} = [1 - P(0)]\mu = \rho\mu = \lambda$.
- Le taux d'utilisation : $\mathbf{U} = [1 - P(0)] = \rho$.
- Le nombre moyen des clients **N** :

$$\mathbf{N} = \rho + \frac{\rho^2(1+CV^2)}{2(1-\rho)}$$

  (Formule de Pollaczeck-Kinchin). avec CV^2 est le cœfficient de variation du temps de service.
- Le temps moyen de séjour **R** :

$$\mathbf{R} = \frac{N}{D} = \frac{1}{\mu} + \frac{\lambda(1+CV^2)}{2\mu^2(1-\rho)}$$

.

Avec CV^2 est le carré du coefficient de variation de la loi de temps de service.

Remarques 2.4.1. *Tous les paramètres de performances moyens d'une file M/G/1 peuvent être déterminés par la simple connaissance des deux premiers moments de la loi de service (*$\mathbf{E}(X)$ *et* $\mathbf{E}(X^2)$*, où le carré de son cœfficient de variation* CV^2*). Cela signifie que deux files M/G/1 de même taux d'arrivée* λ *et ayant les deux premiers moments égaux, ont les mêmes paramètres de performances moyens, même si leurs probabilités stationnaires sont différentes. Il est donc inutile de déterminer les probabilités stationnaires si on ne s'intéresse qu'aux paramètres moyens.*

2.4.5 G/G/1

La file G/G/1 généralise les modèles de files d'attente à un seul serveur (les inter-arrivées et les services sont arbitraires). Cette généralisation rend impossible la détermination des résultats exacts des paramètres de performance pour ce modèle. En effet, un modèle contenant deux lois générales (non Markoviennes) ne peut ni être résolu par la détermination des probabilités transitoires, ni par l'intermédiaire d'une chaîne de Markov induite [12]. Les principaux résultats de cette file sont des bornes inférieures et supérieures qui encadrent le temps moyen d'attente, noté W, dans la file [50] :

$$\frac{\lambda\sigma_X^2 - X(2-\rho)}{2(1-\rho)} \leq W \leq \frac{\lambda(\sigma_T^2 + \sigma_X^2)}{2(1-\rho)} \qquad (1)$$

avec

X **:** le temps de service moyen.

σ_X **:** l'écart type de la variable aléatoire qui décrit le temps de service.

σ_T **:** l'écart type de la variable aléatoire qui décrit les inter-arrivées.

Cette borne peut être décrite d'une façon plus simple si la condition suivante est vérifiée :

$$E[T - t | T > t] \leq \frac{1}{\lambda} \qquad \forall t \geq 0$$

Cette condition est satisfaite pour la plupart des distributions, à l'exception de la distribution hyper-exponentielle. Elle nous permet cependant de réécrire l'inégalité (1) sous la forme :

$$W_{sup} - \frac{1+\rho}{2\lambda} \leq W \leq W_{sup} \qquad (2)$$

avec

$$W_{sup} = \frac{\lambda(\sigma_T^2 + \sigma_X^2)}{2(1-\rho)}$$

En appliquant la formule de Little, on obtient deux bornes pour le nombre moyen de clients N_{buffer} qui attendent le service :

$$\lambda W_{sup} - \frac{1+\rho}{2} \leq N_{buffer} \leq W_{sup}$$

Cette inégalité est très importante. En effet, les deux bornes encadrent de très près le nombre de clients : La différence entre la borne supérieure et la borne inférieure est de $\frac{1+\rho}{2}$ avec $0 \leq \rho \leq 1$ pour une file stable. En d'autres termes, la différence est comprise entre 0 et 1, ce qui représente une bonne approximation du nombre de client qui se trouve dans le buffer N_{buffer}.

Le deuxième résultat intéressant dans le cas d'une file G/G/1 est l'approximation du trafic dense, appelé *heavy traffic* : La distribution de W devient approximativement exponentielle et sa valeur moyenne tend alors à W_{sup} quand $\rho \to 1$. Cela nous ramène à prendre en considération deux types de trafic lorsque on traite une file de type G/G/1 : le trafic au voisinage de la saturation (heavy traffic) lorsque ρ tend vers 1 et le trafic normal ou le bas trafic (low or normal traffic) lorsque ρ est largement inférieur à 1.

2.4.6 G/G/k

La file G/G/k généralise le cas d'une file à plusieurs serveurs (inter-arrivées et services arbitraires). De même que la file G/G/1, il est impossible de déterminer d'une façon exacte les paramètres de performance moyen de cette file. Par ailleurs, des bornes pour les temps moyens d'attente dans la file W_q existent.

$$W_{sup} - \frac{(k-1)X^2}{2kX} \leq W_{G/G/k} \leq \lambda \frac{\sigma_T^2 + \frac{\sigma_X^2}{k} + \frac{(k-1)X^2}{k^2}}{2(1-\frac{\rho}{k})} \qquad (3)$$

avec W_{sup} est la borne supérieure de W pour la file G/G/1 (voir la section 2.4.5).

Remarques 2.4.2. *L'inégalité (3) montre la similitude qui existe entre les bornes déterminées dans le cas de la G/G/1 et de la G/G/k. En effet, le résultat de la G/G/k peut être considéré comme étant l'application de l'inégalité (1) pour un serveur k fois plus rapide avec un temps de service moyen de $\frac{X}{k}$ et une variance de $\frac{\sigma_X^2}{k^2}$.*

Le deuxième résultat important pour cette file est l'approximation pour du trafic dense (heavy traffic). Quand $\frac{\rho}{k}$ tend vers 1, le temps d'attente au régime stationnaire tend vers une variable aléatoire de distribution exponentielle avec une moyenne de :

$$W_{G/G/k} \approx \lambda \frac{\sigma_T^2 + \frac{\sigma_X^2}{k}}{2(1-\frac{\rho}{k})}$$

Les résultats déterminés pour cette file représentent des résultats globaux et valables pour toutes les autres files (M/M/1, M/G/1, M/G/k . . .). Toutefois, ces résultats sont moins précis que les résultats propres à ces files.

2.5 Réseaux des files d'attente et algorithmes de résolution

La résolution des réseaux des files d'attente se base sur des méthodes analytiques qui calculent les probabilités à l'équilibre de ces réseaux, puis déterminent les paramètres de performance moyens, en l'occurrence le temps de réponse moyen, le nombre moyen des clients. . . etc. La détermination des probabilités stationnaires de tous les états possibles d'un réseau constitue le problème central de la théorie des files d'attente [19].

Toutefois, il est possible de résoudre une classe spécifique de réseaux de files d'attente, ayant une structure spéciale permettant d'obtenir la solution sans générer l'espace d'états sous-jacent. Cette classe de réseaux est connue sous le nom de *réseaux de files d'attente à forme produit.*

Définition 2.5.1. *Un réseau est appelé* à forme produit *si et seulement si sa solution au régime stationnaire peut s'écrire sous la forme d'un produit de plusieurs entités dont chacune décrit l'état d'un nœud. Ce type de réseau présente l'avantage de pouvoir calculer sa solution en régime stationnaire sans passer par le régime transitoire, ni la détermination de son espace d'états.*

Un réseau est appelé à forme produit *est également appelé* réseau séparable.

Nous présentons, dans cette section, l'ensemble des algorithmes qui permettent de résoudre les réseaux des files d'attente à forme produit et à forme non produit. Ces algorithmes servent de base pour l'analyse et l'évaluation des performances d'un système donné, effectuées dans un objectif de dimensionnement.

2.5.1 Algorithmes de résolution des réseaux de files d'attente à forme produit

Selon les modèles des files d'attente qui constituent un réseau et selon plusieurs autres paramètres (mono ou multi-classes, ouvert ou fermé), plusieurs algorithmes existent. Ces algorithmes permettent, d'une façon exacte ou approchée, de déterminer les paramètres moyens de performance au régime stationnaire de ces réseaux (temps de réponse moyen, nombre moyen de clients. . .).

Mais avant de donner les solutions exactes et approximatives, regardons la forme des probabilités à l'équilibre d'un réseau à forme produit.

a Analyse du régime permanent

La forme produit varie d'un réseau à un autre. Dans le cas d'un réseau ouvert, mono-classe, de routage probabiliste et dont tous les nœuds sont des files M/M/1, la probabilité stationnaire possède la forme produit suivante :

$$p(n) = \prod_{i=1}^{M} p_i(n_i)$$

où M représente le nombre de nœuds dans le réseau et $p_i(n_i)$ est la probabilité stationnaire d'une file M/M/1 de taux d'arrivée λ_i et de taux de service μ_i, soit :

$$p_i(n_i) = (1 - \frac{\lambda_i}{\mu_i})(\frac{\lambda_i}{\mu_i})^{n_i}$$

Ce résultat est connu sous le nom de *théorème de Jackson* pour les réseaux ouverts. La solution stationnaire a été introduite pour la première fois par [43, 37] pour les réseaux fermés et ouverts. [11] ont étendu ces résultats en déterminant la forme produit des réseaux multi-classes et mixtes, avec différentes disciplines. Soit v_i le taux de visite des clients à la station i ou le nombre moyen de passage d'un client au cours de son séjour dans le système.

Remarques 2.5.1. *Cette forme produit pousse à dire que toutes les stations du réseau sont indépendantes et reçoivent des flux poissoniens. Alors que dans le cas d'un réseau contenant des boucles, les arrivées ne seront plus poissoniens et les stations peuvent être dépendantes. Même dans ces cas, le théorème de Jackson nous confirme que la solution stationnaire d'un tel réseau est la même que si on avait des arrivées poissoniennes et des stations indépendantes [1].*

La solution stationnaire d'un réseau à forme produit est indépendante du type du routage. Elle ne dépend que des taux de visite au niveau de chaque station [12]. Par conséquent, le calcul des paramètres de performances moyens se base uniquement sur les taux de service et le taux de visite de chaque station. En se basant sur le théorème de Jackson, plusieurs travaux [79, 78, 80, 11, 24, 84] ont proposé des algorithmes itératifs exacts ou approximatifs permettant de déterminer les paramètres de performance moyens au régime stationnaire.

b Solutions Exactes

b.1 Algorithme d'analyse par valeur moyenne ou MVA (Mean value analysis) C'est un algorithme récursif, développé initialement par Reiser [78, 80]. Son principe est simple : il consiste à exprimer les paramètres de performance moyens d'un réseau à l'étape où il contient N clients, en fonction de ceux de l'étape de N-1 clients. Il suffit alors d'itérer ces équations en partant des conditions initiales jusqu'à atteindre le nombre total de clients.

Les équations qui relient les paramètres de performance se basent sur deux relations clés : la première est la formule de Little appliquée à tout le réseau et la deuxième consiste dans un théorème appelé *théorème des arrivées* [80, 85].

Théorème 2.5.1. *: Le* théorème des arrivées *consiste à exprimer le temps moyen de séjour d'un client à la station i quand le réseau contient N clients en fonction du nombre moyen de clients à la station i lorsque le réseau contient N-1 clients.*

$$R_i(n) = \frac{1}{\mu_i}(1 + N_i(n-1))$$

avec

- $R_i(n)$ *: Le temps de séjour moyen d'un client dans une station i lorsque le réseau contient n clients.*
- $N_i(n)$ *: Le nombre de client moyen d'une station i lorsque le réseau contient n clients.*

Cette deuxième relation est due au fait qu'un client qui arrive à une station contenant N clients attend, en moyenne, la fin du service du client occupant le serveur, puis les services complets des N-1 autres clients. Sachant que Sevcik et Mitrani [80, 85] ont démontré que pour un réseau à forme produit, le nombre moyen de clients vus par l'arrivée d'un nouveau client est égale au nombre moyen de clients dans la station si un client était supprimé du réseau, d'où la deuxième relation de l'algorithme MVA.

Pour un réseau à M files d'attente et un nombre totale de clients N_{total}, l'algorithme MVA se présente comme suit :

Algorithme 1 « MVA » dans le cas des réseaux mono-classes ouverts à forme produit

Initialisation : $N_i(0)=0$ avec i=1,..,M
Début
Pour n variant de 1 à N_{total} faire
$R_i(n) = \frac{1}{\mu_i}(1 + N_i(n-1))$ avec i=1,..,M
$D(n) = \frac{n}{\sum_{i=1}^{M} v_i R_i(n)}$
$D_i(n) = v_i D(n)$ avec i=1,..,M
$N_i(n) = R_i(n) D_i(n)$ avec i=1,..,M
Fin

Baskett, Chandy, Muntz et Palacios [11] ont étendu cet algorithme aux réseaux multiclasses. Cette extension a été possible grâce au théorème dit « Théorème BCMP », qui donne la forme produit de ce type de réseau. La complexité de l'algorithme MVA est en O(MN), cela veut dire que le temps d'exécution est en fonction du nombre de stations M dans le réseau et du nombre de clients N. Dans le cas d'un réseau de files d'attente avec un grand nombre de clients ou avec un grand nombre de stations, l'algorithme arrive vite à un débordement de mémoire, en particulier dans le cas de stations à plusieurs serveurs. Plusieurs travaux ont cherché des approximations de ces algorithmes, afin de réduire ce temps d'exécution (voir figure 2.5).

c Algorithmes Approximatifs pour les réseaux à forme produit

Lorsque le nombre de clients est très grand, on a recours à des solutions approximatives qui fournissent des résultats approchés, mais avec des temps d'exécution plus rapides que les solutions exactes. Ces solutions peuvent etre classées en trois groupes :

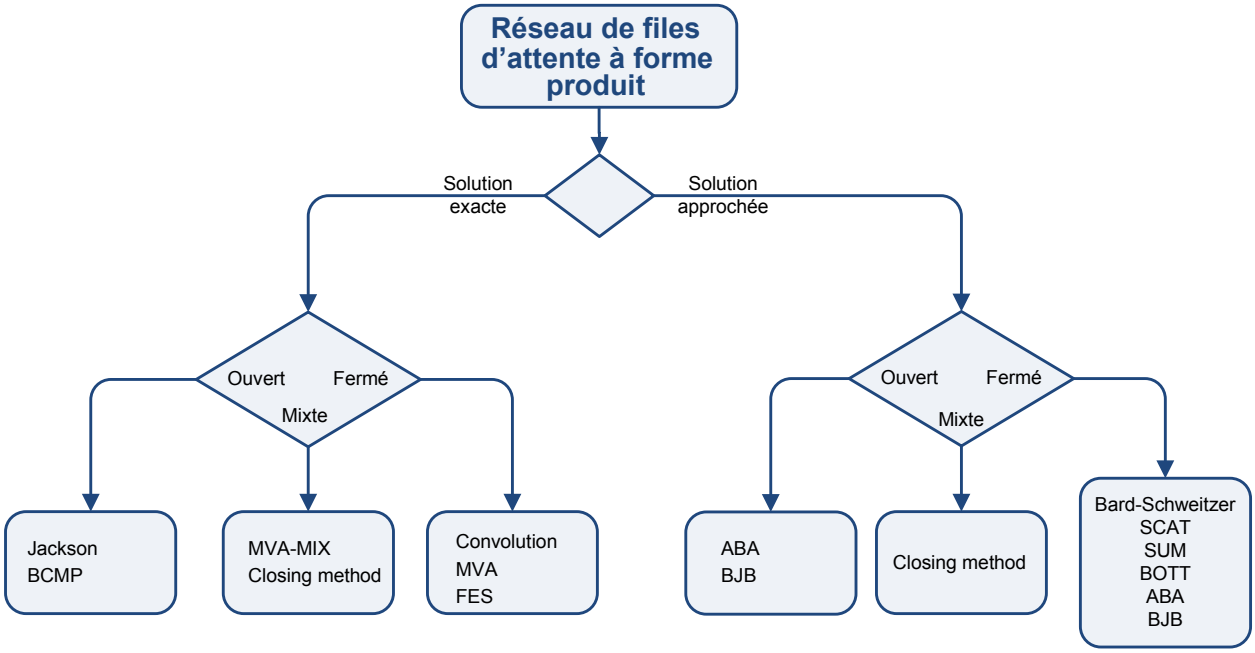

FIGURE 2.5 – Différents algorithmes pour la résolution des réseaux de file d'attente à forme produit

1. Le premier groupe d'algorithmes se base sur l'algorithme MVA, en cherchant à réduire son temps d'exécution et son utilisation mémoire. L'approximation la plus connue est celle de Bard-Schweitzer[84, 10]. Cette approximation consiste à :
   - Estimer, en premier, la longueur des files des différentes stations à l'étape où le réseau contient N clients.
   - Calculer le temps de réponse moyen et le débit moyen en fonction de la valeur estimée par une seule itération de l'algorithme MVA.
   - Réutiliser ces valeurs (temps de réponse moyen et le débit moyen) afin de calculer de nouveau la longueur des files.
   - Calculer l'erreur E entre la première longueur de la file estimée et celle calculée.
   - Réitérer cette procédure jusqu'à avoir à $E < \epsilon$, ϵ préalablement défini.

   Cet algorithme est très facile à implémenter et plus rapide que le MVA exact, surtout lorsqu'on a un nombre important de classes. Le principal inconvénient de cet algorithme est qu'il ne résout pas les réseaux dont les stations contiennent plusieurs serveurs.

   L'algorithme SCAT[72, 24] a étendu le travail de Bard-Schweitze en ajoutant l'estimation de la variation du nombre moyen de clients entre deux itérations successives et en donnant une valeur de ϵ en fonction du nombre de clients dans le réseau. Ces modifications ont permis d'avoir une erreur moins importante par rapport aux paramètres de performances calculés par l'algorithme MVA exact, et de proposer une solution pour le cas de plusieurs serveurs.

   Un étude comparative entre les différentes approximations de l'algorithme MVA a été réalisée dans [99]. Les résultats de cette étude sont résumés dans la figure 2.6. La complexité

Algorithmes	Complexité en temps d'exécution	Complexité en consommation mémoire
MVA exact	$O(MC_l \prod_{c=1}^{C_l}(N_{cl}+1))$	$O(MC_l \prod_{c=1}^{C_l}(N_{cl}+1))$
Bard-Schweitzer	$O(MC_l)$	$O(MC_l)$
SCAT	$O(MC_l^3)$	$O(MC_l^2)$
SCAT amélioré	$O(MC_l^2)$	$O(MC_l^2)$

TABLE 2.1 – Coût d'exécution et consommation mémoire

du temps d'exécution et de la consommation mémoire de ces algorithmes est décrite dans le tableau 2.1.

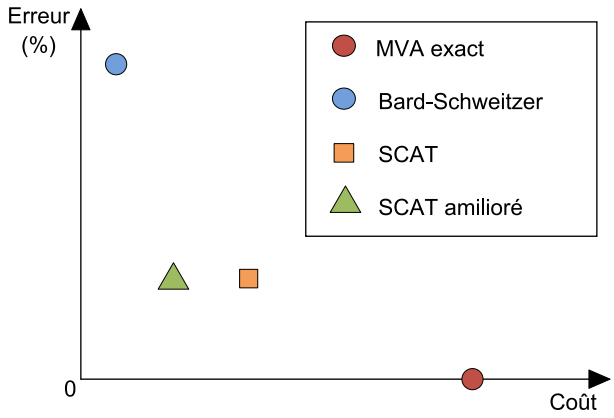

FIGURE 2.6 – Coût d'exécution et taux d'erreurs relatifs aux différentes approximations de l'algorithme MVA

2. Le deuxième groupe est basé sur le principe que le nombre moyen de clients dans chaque station peut être calculé approximativement si on connaît l'utilisation et le débit de cette station. La méthode de sommation SUM[18] estime que le nombre moyen de clients dans chaque station est en fonction de son débit. Cette méthode a permis d'avoir un temps d'exécution plus réduit et une faible consommation mémoire. Mais, le nombre d'itérations de l'algorithme reste important pour un grand nombre de réseaux si on souhaite avoir des résultats dont la précision est acceptable. Afin de réduire le nombre d'itérations de cet algorithme. L'algorithme BOTT[18] propose de bien choisir les conditions initiales. Cette amélioration est connue sous le nom de *bottleneck approximation*, puisqu'elle se base sur les nœuds réalisant la plus grande utilisation (d'où le nom bottleneck ou goulet d'étranglement).

3. Le troisième et le dernier groupe de solutions consiste à trouver des bornes maximales ou minimales pour les indices de performances globaux du réseau. Le schéma 2.6 résume la répartition de ces algorithmes suivant le type de classes de clients (fermé, ouvert ou mixte) et suivant la nature de la solution approchée ou exacte.

2.5.2 Algorithmes de résolution des réseaux de files d'attente à forme non-produit

Malgré le nombre important de solutions pour les réseaux à forme produit, la plupart des réseaux en pratique ne peuvent pas être modélisés par des réseaux à forme produit. D'où la nécessité d'utiliser des réseaux à forme non produit et de pouvoir les résoudre. Plusieurs solutions approximatives pour ce dernier type de réseau existent.

a Méthode basée sur la « Résistance » des réseaux fermés

La solution la plus simple pour la résolution d'un réseau de files d'attente à forme non produit fermé de type -/G/1 et -/G/k consiste à remplacer les lois des temps de service générales des files par des lois exponentielles et de résoudre le réseau avec ses solutions à forme produit. Cette méthode se base sur la propriété de *résistance* (robustness) des réseaux fermés.

Théorème 2.5.2. *La propriété de* résistance *des réseaux fermés est énoncée comme suit : toute modification importante dans les paramètres du système génère des petites variations dans les performances mesurées. Dans le cas des réseaux fermés à forme non produit et de discipline FIFS, la modification des coefficients des variations des lois de temps de service des stations du réseau engendre une faible variation dans les performances calculées.*

Cela veut dire que quelque soit les lois de services des stations d'un réseau fermé, ces derniers peuvent être remplacé par des lois exponentielles sans que cette modification n'engendre pas une grande variation des paramètres moyens du réseau.

Cette propriété a été vérifiée pour 100 réseaux fermés différents et a montré [18] que la variation est de :

– 6% pour les réseaux dont les stations sont mono-serveurs.
– 4% pour les réseaux dont les stations sont multi-serveurs.

Toutefois, cette propriété devient moins précise dans le cas des réseaux multi-classes. Par ailleurs, l'influence du coefficient de variation pour les réseaux ouverts à forme non produit est très grande, en particulier dans le cas des réseaux en tandem sans boucle de retour.

b Méthode de Marie

Cette méthode a été introduite par Marie [64, 63], comme un algorithme itératif applicable pour les réseaux à forme non produit fermés, dont les files sont à un ou plusieurs serveurs, avec des temps de service de distribution arbitraire (loi générale). Elle s'applique en plusieurs étapes. Soit un réseau formé de M stations, munies chacune d'un taux d'arrivée λ_i et un taux de service μ_i.

1. La première étape de l'algorithme consiste à substituer le réseau à forme non produit par un réseau à forme produit ayant la même topologie. Ce réseau de substitution est déterminé en remplaçant chaque loi de service générale d'un nœud par une loi exponentielle et un taux de service $\mu_i(n)$ dépendants de la charge. Les taux $\mu_i(n)$ de ce réseau sont déterminés à partir du réseau original.

2. La deuxième étape consiste à déterminer les taux d'arrivée dépendants de la charge $\lambda_i(n)$, en court-circuitant la station i (i allant de 1 à M) et en remplaçant toutes les autres stations par une station C (voir figure 2.7). Un des algorithmes de résolution des réseaux à forme produit peut être utilisé dans ce cas.

3. La troisième étape consiste à analyser chaque nœud du réseau à forme non produit individuellement et déterminer le taux $\nu_i(n)$ de la station i avec n clients et les probabilités stationnaires $\pi_i(n)$ en utilisant les équations 2.1 et 2.2.

4. La dernière étape consiste à vérifier les conditions d'arrêt 2.3 et 2.4 et calculer les paramètres de performance, Dans le cas où ces conditions ne sont pas vérifiées, on doit calculer les nouveaux taux de service du réseau substitué et réitérer à partir de la deuxième étape.

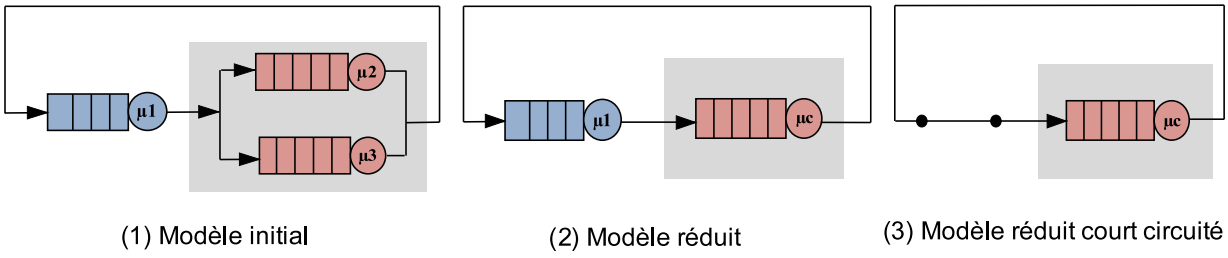

FIGURE 2.7 – Etape1 : Exemple de détermination de $\lambda_i(n)$ pour le réseau substitué

$$\pi_i(n) = \pi_i(0) \prod_{j=0}^{n-1} \frac{\lambda_i(j)}{\nu_i(j+1)} \quad (2.1)$$

avec $v_j$est le taux de visite de la station j, $j = 1, .., K$.

$$\pi_i(0) = \frac{1}{1+\sum_{j=1}^{N} \prod_{n=0}^{j-1} \frac{\lambda_i(j)}{\nu_i(j+1)}} \quad (2.2)$$

$$\left|1 - \sum_{i=1}^{K} \frac{\overline{N}_i}{N}\right| \leq \epsilon \quad (2.3)$$

avec$\overline{N}_i = \sum_{n=1}^{N} n.\pi_i(n)$.

$$\left|\frac{d_j - \frac{1}{K}\sum_{i=1}^{K} d_i}{\frac{1}{K}\sum_{i=1}^{K} d_i}\right| \leq \epsilon \quad (2.4)$$

avec
- $d_j$est le débit normalisé de la station j : $d_j = \frac{\lambda_j}{v_j} = \frac{1}{v_j}\left(\sum_{n=1}^{N} \pi_j(n).\nu_j(n)\right)$.
- ϵ est la tolérance variant entre 10^{-3} et 10^{-4}.

(2.5)

Dans la majorité des cas, la méthode de Marie donne une précision très élevée [18]. Dans ce qui suit nous allons présenter une méthode permettant de résoudre les réseaux de file d'attente à forme non produit ouvert.

c La Méthode de décomposition pour les réseaux à forme non produit ouverts

La méthode de décomposition [52] est une solution approximative pour l'analyse des réseaux ouverts à forme non produit mono ou multi-classes, avec des stations mono ou multiserveurs, dont les lois d'inter-arrivées et de service sont distribuées arbitrairement et la discipline de service est FIFS. Elle est formée de deux étapes majeures

1. La première étape consiste à calculer le taux d'arrivée et l'utilisation de chaque station.
2. La deuxième étape calcule le coefficient de variation de la distribution d'inter-arrivée de chaque nœud. Ce calcul est fait itérativement suivant 3 phases :
   (a) *PHASE 1 : Composition.* Plusieurs processus d'arrivée sont composés pour former le processus d'entrée du nœud i (voir 2.8). Le taux d'arrivée généré est la somme des taux d'arrivées des autres processus d'arrivée. Quant au calcul du coefficient de variation, plusieurs auteurs ont proposé différentes approximations [53, 17, 26, 75, 100, 101, 34].
   (b) *PHASE 2 : Traversée.* Le coefficient de variation des temps d'inter-départ CvD dépend du coefficient de variation du temps d'inter-arrivée CvA et du temps de service CvB. Ce calcul aussi a fait l'objet de plusieurs propositions données par les travaux cités précédemment.

Algorithme 2 Algorithme de Marie

ETAPE 1 : Initialisation

- Substituer le réseau à forme non produit par un réseau à forme produit équivalent en remplaçant chaque loi de service général par la loi exponentielle de paramètre $\mu_i(n)$.
- Calculer $\mu_i(n)$.
- Dans le cas de plusieurs serveurs (k_i) ayant un taux de service $\mu_i(n)$, remplacer les serveurs par 1 seul serveur de taux de service :

$$\mu_i(n) = \begin{cases} 0 & si \quad n = 0 \\ min(n, \mathrm{m}_i).\mu_i & si \quad n > 0 \end{cases}$$

ETAPE 2 : Détermination des $\lambda_i(n)$

- Utiliser le réseau de substitution et une solution de résolution des réseaux FP (MVA ou l'algorithme de convolution) pour déterminer les $\lambda_i(n)$, i=1,..,M et n=0,..,N.
- Pour n=N, on a $\lambda_i(n) = 0$.

ETAPE 3 : Analyse séparée des nœuds

Déterminer les taux $\nu_i(n)$ et les probabilités $\pi_i(n)$ en utilisant :
- les équations 2.1 et 2.2 pour les $\pi_i(n)$.
- le type des nœuds [63] pour les $\nu_i(n)$.

ETAPE 4 : Conditions d'arrêt

Vérifier les deux conditions d'arrêt 2.3 et 2.4 :
Si (2.3 et 2.4 sont vérifiées) **alors**
Déterminer les paramètres de performance N,U,D ...
Sinon
$\mu_i(n) = \nu_i(n)$
Aller à l'étape2
Fin Si

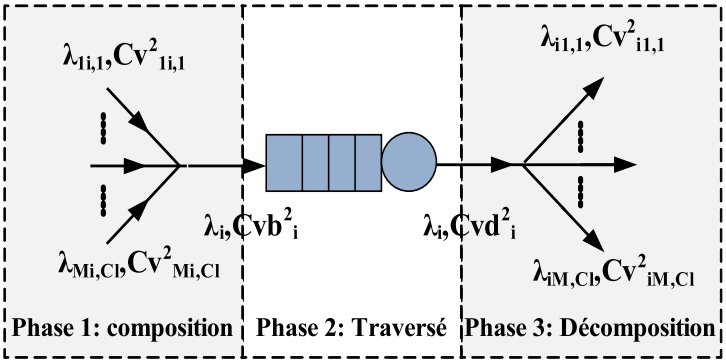

FIGURE 2.8 – Calcul du coefficient de variation par la méthode de décomposition pour les réseaux ouverts

(c) *PHASE 3 : Composition.* Pour la phase de décomposition, tous les auteurs ont utilisé la formule suivante :

$$C^2_{ij,Cl} = 1 + p_{ij,Cl}.(C_{Di} - 1)$$

3. En utilisant les résultats de la file G/G/1 et la file G/G/k, la troisième étape calcule les paramètres de performance moyens N et W.

d Autres solutions

Beaucoup d'autres travaux ont proposé des solutions approximatives pour les réseaux à forme non produit dans des cas spécifiques. Le schéma 2.9 montre la répartition des différentes solutions suivant leurs types de classes de clients (ouverts, fermés ou mixtes), la distribution du temps de service non exponentielle, la discipline avec priorité [10, 33] ou FIFS, ou des temps de service variable [23, 104].

2.6 Simulation des réseaux de files d'attente

Une simulation est une expérience visant à déterminer les caractéristiques de ce système de manière empirique [18]. Il s'agit d'une méthode permettant d'imiter ou de reproduire le comportement d'un système à travers le temps, dans le but de tirer des conclusions concernant le comportement dynamique du système réel. Parmi les types de simulation, la plus utilisée est la simulation à évènements discrets.

Une simulation à évènements discrets [105, 55] consiste à concevoir un modèle d'un système réel ou théorique, en reproduisant son évolution pas à pas, puis en analysant statistiquement les résultats d'exécution. L'état actuel du système physique est représenté par des variables

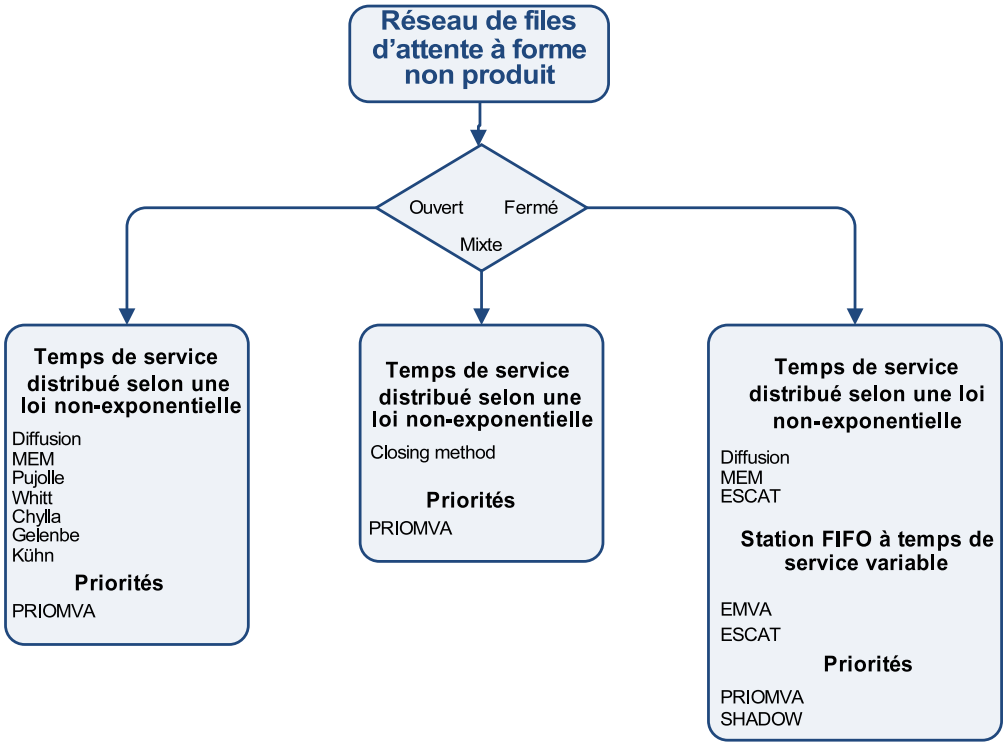

FIGURE 2.9 – Différents algorithmes pour la résolution des réseaux de file d'attente à forme non produit

d'état. Ces variables sont discrètes. Leurs valeurs constituent l'espace d'états du système. Par exemple, dans un réseau de communication, les variables d'état désignent le nombre de paquets en attente d'être traités au niveau des nœuds du réseau. L'espace d'états est dénombrable. Selon la définition de l'espace d'états, chaque changement d'un état (évènement) se produit de manière discrète à travers le temps, à des instants t_i dits « *dates d'événements* ».

La simulation à évènements discrets s'applique à tout système discret, c'est à dire dont l'évolution à travers le temps est accompli de façon discrète. Cette définition couvre une très large gamme de systèmes. Toutefois, elle est généralement complexe à développer et exige un temps de calcul et des ressources (CPU, disques, mémoire, ...) importantes.

Les concepteurs ont généralement recours à la simulation, lorsqu'aucune des méthodes analytiques précédemment présentées n'est applicable. La précision des résultats de la simulation dépend du nombre de clients simulés. Plus le nombre de clients simulés n'est grand, plus les résultats obtenus sont précis, ce qui implique un temps énorme de traitement. Dans notre cas, nous allons utiliser la simulation comme dernier recours pour l'analyse des performances d'un système et en particulier son dimensionnement.

2.7 Conclusion

Ce chapitre a présenté les différentes notions liées aux réseaux de files d'attente et leurs méthodes de résolution.

La première partie du chapitre a porté sur les principales définitions, puis les principaux résultats s'appliquant à l'analyse des files simples. Ces résultats sont essentiellement utilisés dans le chapitre 6, afin de modéliser un système ou un sous-système vu comme une boîte noire.

La deuxième partie de ce chapitre a présenté les différents algorithmes de résolution d'un réseau de file d'attente, suivant ses propriétés. Ces algorithmes sont utilisés par la suite pour le dimensionnement et l'analyse de performances d'un système sous test.

Le prochain chapitre se concentrera sur les différents travaux et approches proposées dans la littérature pour le dimensionnement des systèmes répartis en utilisant les réseaux de files d'attente.

Chapitre 3

Modélisation et dimensionnement

3.1 Introduction

Avant de commencer la deuxième partie de ce manuscrit qui présentera notre approche, on consacre ce chapitre pour présenter les différents travaux existants de modélisation, d'expérimentation automatique et de dimensionnement. Tel que défini dans les chapitres précédents, on a opté pour l'utilisation des réseaux de file d'attente comme modèle. On a choisi un niveau de détails de la modélisation correspondant à une modélisation de point de vue composants. Cela nous permettra de détecter avec précision le ou les composants responsables de la dégradation des performances et permettra de remédier à ce problème au moindre coût. Ce chapitre commence par présenter les différents travaux existant pour la modélisation des systèmes répartis. une deuxième section sera dédiée aux travaux existants de l'autonomic-computing et l'expérimentation. Une troisième section sera consacrée aux travaux de dimensionnement et aux solutions de détection des goulets d'étranglement. Finalement, on présente le positionnement de notre approche par rapport à l'existant.

3.2 Modélisation

Beaucoup de travaux ont modélisé les systèmes répartis par des files d'attente. Ces travaux peuvent être classés selon leurs natures : les distributions utilisées pour modéliser les inter-arrivées et le service, ou selon la structure du modèle : modèle avec une file d'attente unique ou avec un réseau de file d'attente. Un tour d'horizon de ces travaux est présenté dans la suite de cette section en prenant en considération ces deux classifications.

Modélisation par une seule file d'attente : Le modèle M/M/1 est le modèle le plus utilisé dans les études des performances des systèmes répartis vu sa simplicité et les solutions exactes de ses paramètres de performance moyens. Ce modèle a été utilisé pour modéliser des systèmes client-serveur [67], des parties des systèmes distribués (serveur Web, serveur base de données) ou des systèmes à architecture multi-tiers [66]. L'hypothèse poissonnienne des arrivées a été vérifiée et largement acceptée dans différents domaines y compris dans la communauté des systèmes distribués. Pour les distributions de services, beaucoup de travaux ont remis en cause cette hypothèse et ont utilisé des distributions plus générales vu l'inadéquation de cette hypothèse pour les nouveaux systèmes distribués qui sont plus complexes et plus répartis. Parmi ces travaux, [46, 76] ont utilisé des modèles avec une distribution de service générale en se basant sur les résultats de la file M/G/1 (voir chapitre 1) pour évaluer les performances des systèmes multi-tiers ou pour limiter l'accès à ces systèmes afin d'éviter leurs saturations et garantir la qualité de service souhaitée.

La plupart de ces travaux supposent la connaissance préalable des distributions d'arrivées, de service et le nombre de serveurs. D'autre part, quelques travaux [47, 103, **?**] ont considéré le système comme une boîte noire et ont cherché à identifier les modèles sans aucune connaissance sur le fonctionnement interne des systèmes. Parmi ces travaux, Begin et al. [**?**] proposent une méthode de modélisation HLM (High Level Modeling) qui se base uniquement sur des mesures de performances. Cette Méthode cherche à produire un modèle simple et de le calibrer afin de reproduire au mieux les mesures (voir figure 3.1). Les systèmes à modéliser sont ainsi considérés comme des boîtes noires et par conséquent tout système disposant d'un jeu de mesures peut utiliser cette méthode. La modélisation se fait en trois étapes :

- Une première étape pour définir les modèles de file d'attente génériques (briques) à utiliser. Ces briques peuvent être des files d'attente élémentaires (M/M/c, M/G/1), ou des modèles imbriqués pouvant décrire des comportements plus complexes.
- Une deuxième étape de calibrage des modèles qui consiste à déterminer l'ensemble des paramètres associés à chacun des modèles génériques pour les approcher des mesures.
- Et une dernière étape qui consiste à sélectionner le modèle lauréat minimisant la distance entre le modèle et les mesures. Cette méthode semble intéressante puisqu'elle se base que sur des mesures de performances et ne suppose pas un modèle prédéfini. Ces deux critères rendent cette méthode automatisable et permet de modéliser un large spectre de systèmes.

La modélisation avec une seule file d'attente reste insuffisante pour des objectifs de dimensionnement des systèmes distribués puisqu'elle ne permet pas la localisation des parties responsables des problèmes de performances. Néanmoins, les méthodes d'identification décrites dans ces travaux peuvent être réutilisées et adaptées pour modéliser une partie du système et produire un modèle de réseau de files d'attente.

Modélisation par un réseau de files d'attente : Dans leur livre [66], Menascé et al. décrivent une méthode de modélisation de systèmes avec un niveau composant (*Component-Level Performance Models*) en utilisant trois types de files élémentaires pour former des réseaux de files d'attente. Chaque file correspond à un type de ressource (ressource dépendant de la charge, ressource indépendant de la charge et une ressource de délai). La structure du réseau est basé sur la connaissance exacte des mécanismes internes du système. A titre d'exemple, un serveur base

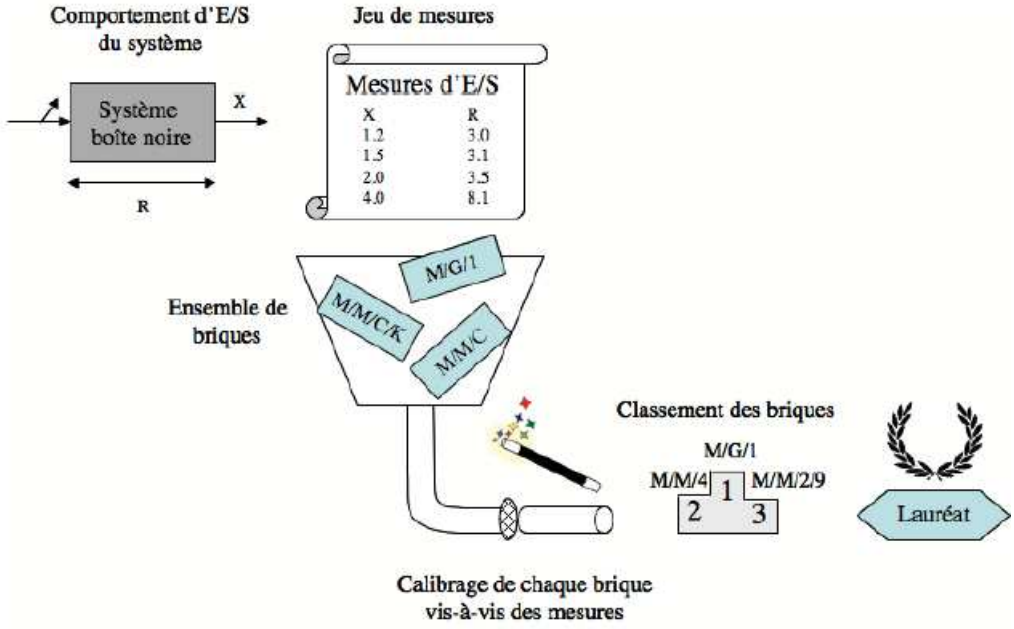

FIGURE 3.1 – Fonctionnement général de la méthode HLM

de données pourra être modélisé par un réseau formé de deux files d'attente dont la CPU est caractérisée par une file dépendante de la charge et le disque par une file d'attente indépendante de la charge.

D'autres travaux ont modélisé avec des réseaux de file d'attente en utilisant une approche boîte noire. Chaque file modélise une partie du système tournant sur une ou plusieurs machines physiques séparées. Parmi ces travaux, Urgaonkar et al. [96] ont modélisé une application 3-tiers avec un réseau en tandem composé de trois files G/G/1 dont chaque file correspond à un tiers. Ce travail est très intéressant puisqu'il impose peu de contraintes sur les distributions des inter-arrivées et de services. Étant donné qu'on ne dispose pas des résultats exactes pour cette file, les auteurs utilisent des bornes majorantes pour encadrer les temps de réponse de bout en bout et garantir une qualité de service liée à ce temps.

Plusieurs travaux [61, 4, 65] ont utilisé des boucles de contrôle et d'asservissement pour le calibrage des paramètres de configuration et l'optimisation des performances du système. Ces travaux se basent généralement sur des modèles analytiques qui supposent une connaissance complète ou partielle du comportement du système. Ces modèles sont mis à jour après chaque itération de contrôle. Tarek F. Abdelzaher et al. [4] ont présenté une approche théorique et une architecture de gestion de qualité de service d'un serveur web apache utilisant une boucle de contrôle (voir figure 3.2). Le modèle utilisé est une fonction de transfert lineaire (time-varying linear transfer function) pour le contrôle de l'utilisation du serveur web. Ying Lu, Abdelzaher et al. [61] ont montré expérimentalement l'efficacité de la combinaison d'une boucle de contrôle avec un modèle de prédiction de file d'attente M/M/1 pour le contrôle d'un temps de réponse d'un serveur web (appache/Http1.1).

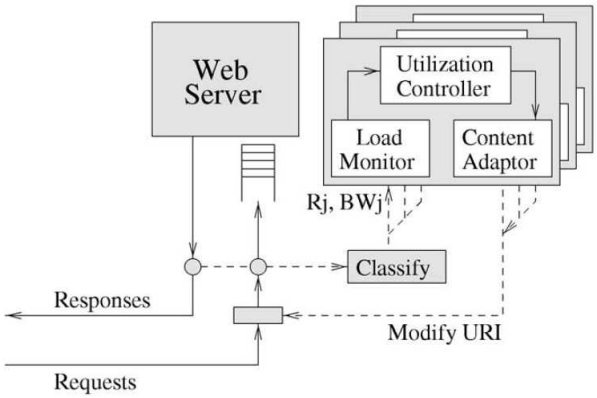

FIGURE 3.2 – Architecture pour le contrôle de l'utilisation d'un serveur apache

Menascé et al. [65] ont fourni une méthode qui permet de contrôler et assurer une qualité de service d'un site e-commerce 3-tiers. Le contrôle porte sur l'optimisation dynamique et incrémentale des paramètres de configuration (nombre de threads, nombre de connexions maximales, etc). Trois paramètres de qualité de service sont pris en charge : temps de réponse, le débit et la probabilité de rejet. La figure 3.3.a décrit l'architecture du contrôleur utilisé. Ce contrôleur permet de caractériser la charge en entrée du site et évaluer la qualité de service par rapport à un paramètre de Qos fixé. Ces deux observations permettent d'adapter d'une façon incrémentale les paramètres de configuration en se basant sur le modèle de réseau de file d'attente décrivant le site e-commerce (voir figure 3.3.b). Le modèle utilisé est une combinaison d'une vue système et une vue composant comme décrite dans [66].

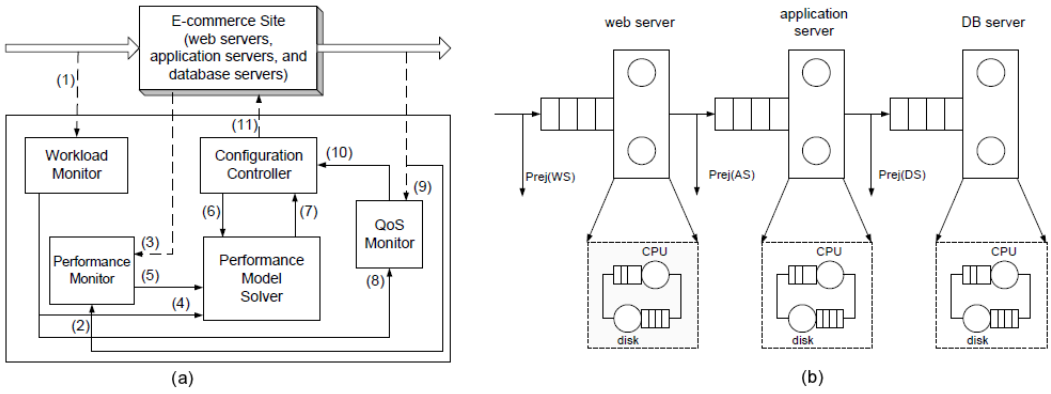

FIGURE 3.3 – (a) Architecture du contrôleur de la QoS de [65], (b) Modèle de performance du e-commerce utilisé

Dans [68], Menascé et al. se sont focalisés sur l'utilisation à la volée d'un modèle de performance analytique pour la conception des systèmes autonomiques. Différentes alternatives de conception du contrôleur ont été présentées et discutées. Le travail a montré aussi l'importance de la prédiction de la charge de travail dans la conception de ces systèmes. Woodside, Zheng and Litoiu [106, 102] ont étudié l'optimisation dynamique des paramètres des modèles de performances pour l'autonomic computing. Les auteurs ont utilisé différents modèles de per-

formances : réseau de files d'attente en couches (*Layered Queuing Networks*), réseau de pétri, etc. Ils ont appliqué une adaptation des filtres de kalman pour l'optimisation automatique des paramètres de modèle de performances (temps de réponse et utilisation) à partir des mesures. Les auteurs ont montré aussi la convergence de l'estimateur du filtre de Kalman avec des conditions facilement vérifiables.

3.3 Dimensionnement

Depuis le développement du World Wide Web, beaucoup des travaux se sont intéressés à la caractérisation du trafic web et à l'optimisation des paramètres internes des serveurs notamment avec les techniques d'optimisation des caches web niveau client et serveurs. Depuis, les travaux de dimensionnement et de configuration n'ont pas cessé de répondre au nouveaux besoins de dimensionnement liés à la complexité croissante des nouvelles architectures et aux nouvelles contraintes de qualité de service. Plusieurs travaux ont adapté les techniques de dimensionnement au contexte autonomique [59, 96, 7]. L'objectif principal de ces travaux est d'ajuster dynamiquement la configuration du système à la charge en entrée ou inversement afin de garantir la qualité de service souhaitée. Les premiers travaux ont utilisé un ajustement qui consiste à contrôler la charge en entrée du système à travers des mécanismes de contrôle de flux pour éviter la contention. Ces types d'ajustement ne peuvent pas être considérés comme des solutions de dimensionnement. Néanmoins, ces travaux ont largement servi aux travaux de dimensionnement surtout que les parties de caractérisation de la charge en entrée et d'observation des performances du système restent identiques. Les particularités des solutions de dimensionnement résident dans la phase de réaction qui cherche à reconfigurer les systèmes à travers des mécanismes de réplications ou à travers les modifications des paramètres système afin d'accepter la charge en entrée sans autant la modifier. Afin de décider de la nécessité de la réplication, toute procédure de dimensionnement doit calibrer en premier le modèle du système avec les mesures observées. Ensuite ce modèle sera résolu pour comparer les performances du système au SLO. En cas de violation du SLO, une reconfiguration est nécessaire pour réserver plus de ressources. Dans le cas d'un dimensionnement dynamique, la reconfiguration doit être également générée et décidée automatiquement. Dans le cas général, plusieurs reconfigurations sont à tester en solvant le modèle associé avant de décider de la reconfiguration solution qui résout définitivement le problème. Étant donnés les modèles des systèmes répartis présentés dans la section précédente, on présente dans la suite la phase de résolution et de vérification du SLO de chaque type de modèle.

Pour les modèles à une seule file d'attente[46, 76, 66, 67], la résolution se réduit au calcul des paramètres moyens exacts ou approchés conformément aux résultats du chapitre 2. En cas de violation du SLO, la réplication concernera tous les systèmes puisque le modèle utilisé ne permet pas de détecter avec précision la partie du système responsable des dégradations des performances.

Pour les modèles utilisant les réseaux de file d'attente, la recherche d'un algorithme de résolution dépend en premier des types de files d'attente et de la structure du réseau. Plusieurs

travaux [6, 7, 66] utilisent l'algorithme MVA pour dimensionner. Cela signifie qu'on modélise le système par des réseaux à forme produit fermé. Arnaud et al. [6, 7] proposent un middleware d'approvisionnement automatique (Moka) pour des architectures multi-tiers utilisant une adaptation de l'algorithme MVA.

Pour dimensionner dynamiquement leur application 3-tiers, Urgaonkar et al. [96] ont utilisé leur modèle en réseau en tandem de G/G/1 présenté dans la section précédente. Les boîtes étant en série donc le temps de réponse n'est autre que la somme des temps passés dans chaque tiers. Les auteurs utilisent alors la borne supérieure de l'approximation du trafic dense (*heavy traffic*) du temps d'attente de la file G/G/1 pour encadrer le taux d'arrivée λ (voir la figure 2.4.5) et déterminer le nombre de serveurs nécessaires (et par conséquent le nombre de réplications) pour chaque tiers pour traiter le taux d'arrivée. De même, Besides et Litoiu [59] montrent comment déterminer des bornes inférieures et supérieures des paramètres de performances (temps de réponse et utilisation). Ils fournissent par la suite une méthode se basant sur la borne supérieure de ces paramètres et sur l'algorithme MVA pour déterminer le nombre de noeud à approvisionner. Ce travail fournit également une méthode de détection des goulets d'étranglement qui sera présentée par la suite.

3.4 Détection des goulets d'étranglement

Plusieurs techniques ont fait l'objet des travaux de recherche des goulets d'étranglements. On exposera trois techniques parmi les techniques les plus utilisées et on discutera de leurs efficacitées et leur adaptation par rapport à notre contexte.

3.4.1 Recheche avec les techniques d'apprentissage

Dans le cadre du projet ELBA (Georgia Tech and HP Labs) [93, 74] sur la configuration et le déploiement automatique des applications multi-tiers, Jason Parekh et al.[45] ont mis en place une méthode de détection des goulets d'étranglement, en fusionnant des techniques d'apprentissage et d'analyse de performance. Etant donnée que la détection de la partie du système responsable du goulet d'étranglement est un problème difficile, les auteurs se fixent comme objectifs de déterminer plutôt les mesures responsables de ce goulet d'étranglement. Le processus de détection se fait alors en trois phases :

- La première est la phase de préparation des instances d'apprentissage en variant la charge injectée.
- La deuxième est l'apprentissage du classificateur avec les différentes instances de la première phase pour générer un modèle qui définit la corrélation qui existe entre les mesures et la violation du SLO.
- La troisième phase est l'identification des goulets d'étranglement par le classificateur en comparant l'instance réelle donnée au modèle généré. Il détermine, de cette façon, si le système sous test à un moment donné présente un goulet d'étranglement ou non.

L'idée est de créer plusieurs instances d'apprentissage qui sont des couples formés par le niveau de charge injectée au système et par la réponse du système exprimée sous forme de pourcentage de violation du SLO. Quelques instances seront truquées de façon à provoquer des goulets d'étranglement et elles serviront à faire apprendre aux différents classificateurs. Une fois que la phase d'apprentissage est terminée, chaque classificateur génère son propre modèle. Il l'utilisera pour détecter les éventuels goulets d'étranglement dans les mesures collectées de l'expérience réelle.

3.4.2 Recherche statistique des goulets d'étranglements

Malkowski et al.[62] ont adapté les travaux réalisés au projet Elba en éliminant les techniques d'apprentissage et en les remplaçant par une recherche statistique du goulet d'étranglement. Le processus reste le même, seule la méthode de détermination de la mesure responsable du goulet change. L'idée est alors de chercher statistiquement un niveau de charge C qui sépare l'ensemble des charges injectées en deux intervalles disjoints. Un premier qui satisfait le SLO. Un deuxième intervalle qui viole considérablement le SLO. Pour déterminer ce point C, les auteurs proposent un algorithme itératif qui cherche une approximation de ce point par une simple heuristique. Ils commencent alors avec une charge suffisamment faible et ils ajoutent itérativement de la charge. Après chaque itération, l'algorithme cherche une approximation d'un intervalle de confiance de 95% du SLO. Lorsque la borne inférieure de l'intervalle de confiance devient plus petite que 90%, l'algorithme s'arrête et la charge de sortie présente dans ce cas une approximation de la charge C. Toute charge supérieure à cette valeur présente forcement un goulet d'étranglement puisqu'elle viole le SLO.

3.4.3 Recherche avec les mesures d'utilisation

Serazzi, Balbo et Litoiu [8, 9, 59] considèrent un goulet d'étranglement comme étant une ressource dont l'utilisation est égale à 1. c'est-à-dire quand elle est occupée à 100%. Ils classifient les goulets d'étranglement en deux catégories :

- La première est appelée «*Natural bottleneck*» lorsque le goulet est causé par une seule ressource.
- La deuxième est appelée «*Joint bottlenecks*» lorsque plusieurs ressources arrivent simultanément à leurs limites d'utilisation.

Les auteurs montrent aussi que l'ensemble des goulets d'étranglement peuvent être déterminés en solvant une équation linéaire [59] en fonction de l'utilisation par ressource et par classe de client et la demande de ressource par classe de client. La résolution de cette équation permet de déterminer ce qu'ils appellent zone de saturation et qui regroupe tous les éventuels goulets d'étranglement.

3.5 Positionnement par rapport à l'existant

Modélisation : Le choix d'un modèle est une partie principale et décisive du processus de l'évaluation de performance d'un système. Les premiers travaux présentés ont modélisé le système avec une seule file d'attente. Ces travaux avaient comme objectif principal de reconfigurer des serveurs web en modifiant uniquement quelques paramètres internes (nombre max de clients, nombre de threads, etc) ou de reconfigurer un serveur avec un asservissement par rapport au temps de réponse de bout en bout d'une architecture 3-tiers. Ces travaux sont donc inappropriés à notre contexte puisqu'ils ne permettent pas de détecter et de localiser finement un goulet d'étranglement. Un modèle de réseau des files d'attente est donc nécessaire. Il permet d'éviter un redimensionnement démesuré. Néanmoins, tous ces modèles peuvent être réutilisés pour modéliser les composants formant le système.

Les travaux de Begin et al. sont intéressants puisqu'ils ne choisissent pas les distributions des inter-arrivées et des services d'une façon préalable. Ils se lancent alors dans un processus de sélection d'un modèle parmi plusieurs. Cette ligne de réflexion semble prometteuse vu la complexité des systèmes récents au point où même la modélisation de la charge d'arrivée par un processus de poisson peut être remise en cause dans plusieurs cas en faveur des processus de type queue lourde. Notre travail reprend en partie l'idée de chercher un modèle optimal parmi plusieurs, tout en proposant une méthode d'exploration et d'identification des modèles des inter-arrivées et des services plus fine (voir chapitre 6). Concernant le travail de Menascé et al [65], l'idée de choisir un modèle qui combine une vue système (modélisation boîte noire) et une vue composant (modèle boîte blanche) est intéressante. En effet, ce modèle permet de profiter de la simplicité du modèle de vue système et en cas de besoin d'explorer les détails du modèle vue composant et aidera la détection et la localisation du goulet d'étranglement. Toutefois, notre travail se veut générique et permettant de modéliser tout système réparti, nous ne pouvons nous inscrire que dans le cadre d'une modélisation de type boîte noire ou boîte grise et qui se basent uniquement sur des mesures. Notre approche ressemble au modèle proposé par Urgonkar et utilise une modélisation en boîte grise générant des modèles de réseau des files d'attente adapté aux systèmes répartis et à leur dimensionnement. Cependant, l'estimation et le calibrage du modèle proposés par Menascé et par Urgonkar se basent en partie sur des données internes chose qu'on pourra pas faire dans notre cas.

Automic computing et boucle de contrôle : L'utilisation de la boucle de contrôle et des concepts de l'autonomic computing dans un contexte de dimensionnement dynamique est inévitable. Tous les travaux ont montré l'efficacité de l'utilisation de cette boucle pour le maintien du modèle et du système à jour. Menascé et al. ont proposé une solution de contrôle utilisant un modèle de réseau de file d'attente qui permet de contrôler la Qos d'une architecture trois tiers. On propose d'adapter cette solution à notre modèle. On propose également d'utiliser cette boucle dans notre approche d'identification du modèle et d'expérimentation. La mise en place de cette boucle de contrôle est décrite plus en détail dans le chapitre 5. D'un autre côté, l'utilisation de Zheng et Litoiu des filtres de Kalman est une bonne solution vue que l'estimation des paramètres est faite d'une façon incrémentale qui correspond parfaitement à processus itératif de la boucle de contrôle. Néanmoins, cette solution a été rejetée pour des problèmes de conver-

gence. En effet, l'estimateur ne garantit pas une convergence pour tous les modèles. Ce qui complique la tâche puisque dans notre approche d'identification on ne connait pas à l'avance le modèle généré et par conséquent on ne peut garantir l'utilisabilité de cet estimateur.

Dimensionnement : Pour les travaux de dimensionnement et de résolution de file d'attente et contrairement à tous les autres travaux, on ne pourra pas fixé un algorithme de résolution à l'avance. En effet, la nature du réseau de file d'attente généré ne dépend que de la phase d'identification. Notre résolution se fera en se basant sur un ensemble d'algorithme de résolution ou de simulation permettant de résoudre un grand nombre de modèles. Dans le cas où plusieurs algorithmes sont disponibles, la sélection sera faite en fonction de la précision et de la rapidité de l'exécution de l'algorithme.

Recherche des goulets d'étranglement : Le projet Elba utilise les techniques d'apprentissage pour l'analyse de performances et la détection des goulets d'étranglement. Ces travaux permettent une détection comparable aux autres techniques habituelles d'analyse de performances de point de vu précision et rapidité de détection. Toutefois, ces techniques se basent sur une phase d'apprentissage très coûteuse en temps surtout lorsqu'un niveau de précision est exigé. De plus, les modèles générés par le classificateur ne sont pas généralement interprétables, donc nous ne pouvons pas l'analyser ni déduire d'autres informations sur les performances. La seule information que nous pouvons avoir d'un tel modèle est alors l'existence d'un goulet d'étranglement dans une instance de mesure fournie.

les travaux de recherche avec des techniques statistiques ont pu éliminer la partie d'apprentissage et ont réduit de cette façon la complexité du modèle utilisé par les classificateurs. Mais cette méthode garde encore beaucoup d'inconvénients. Nous pouvons citer :

1. le nombre d'expériences, qu'il faut faire pour générer le modèle et déterminer l'estimation du point limite *C*, est très grand.
2. l'estimation du point limite *C* est propre à la configuration courante. Ainsi, en cas de changement d'une configuration, l'expert doit refaire tout le travail pour déterminer les nouvelles statistiques et donc la nouvelle valeur de *C*.

Le chapitre 5 indique notre approche expérimentale de détection de goulets d'étranglement qui se base en partie sur l'estimation de l'utilisation. Cela permet d'éviter toutes les solutions nécessitant des phases d'étude préalable statistique ou d'apprentissage.

3.6 Conclusion

Ce chapitre a présenté un tour d'horizon des principaux travaux de modélisation, de dimensionnement et d'autonomic computing. Plusieurs concepts de ces travaux vont être adoptés dans notre méthodologie de dimensionnement. Dans la plupart des travaux de modélisation et de dimensionnement, les auteurs choisissent un modèle par défaut pour représenter les différentes parties de leur système (M/M/1, M/G/1 ...). Ce choix sera très coûteux dans le cas où le modèle ne convient pas à une partie du système. Notre solution cherche en premier à identifier la

famille modèle acceptable pour tous les composants du système. Cette identification nous permettra d'avoir un modèle plus précis et qui prend en considération le comportement réel de nos boîtes. Suivant les modèles des différentes boîtes identifiées, on choisira l'algorithme adéquat pour la résolution du modèle.

Deuxième partie

Vers un dimensionnement automatique

Chapitre 4

Méthode de dimensionnement

4.1 Introduction

Nous avons vu, dans la première partie de ce manuscrit, des différents travaux qui montrent la complexité du processus suivi pour réaliser un dimensionnement d'une application répartie. Cette complexité est présente dans les différentes étapes du processus d'évaluation de performance à savoir :

- le test de performance,
- la collecte des données,
- la modélisation, et
- la résolution du modèle obtenu.

Dans cette deuxième partie, nous développons une approche de dimensionnement dynamique en considérant cette complexité. Notre approche se base sur l'injection de charge pour identifier automatiquement les modèles théoriques des composants d'un système. Elle fournit à l'administrateur plusieurs alternatives résolvant le problème de dimensionnement, ainsi que des éléments d'aide à la décision permettant de garantir la fiabilité des solutions obtenues. Suivant les objectifs visés de l'administrateur, il devra effectuer des choix stratégiques pour assurer une solution adéquate à son problème. Nous montrons à la fin de cette partie comment notre approche de dimensionnement peut être aisément étendue pour réaliser un dimensionnement autonome.

Ce premier chapitre présente une description globale et une vue d'ensemble des différentes étapes de notre approche, qui seront détaillées dans les chapitres suivants. La deuxième partie de ce chapitre définira et expliquera les problèmes de décomposition et d'isolation qui sont des phases préparatoires pour notre approche mais qui ne seront pas développés dans le reste des chapitres.

4.2 Vue d'ensemble

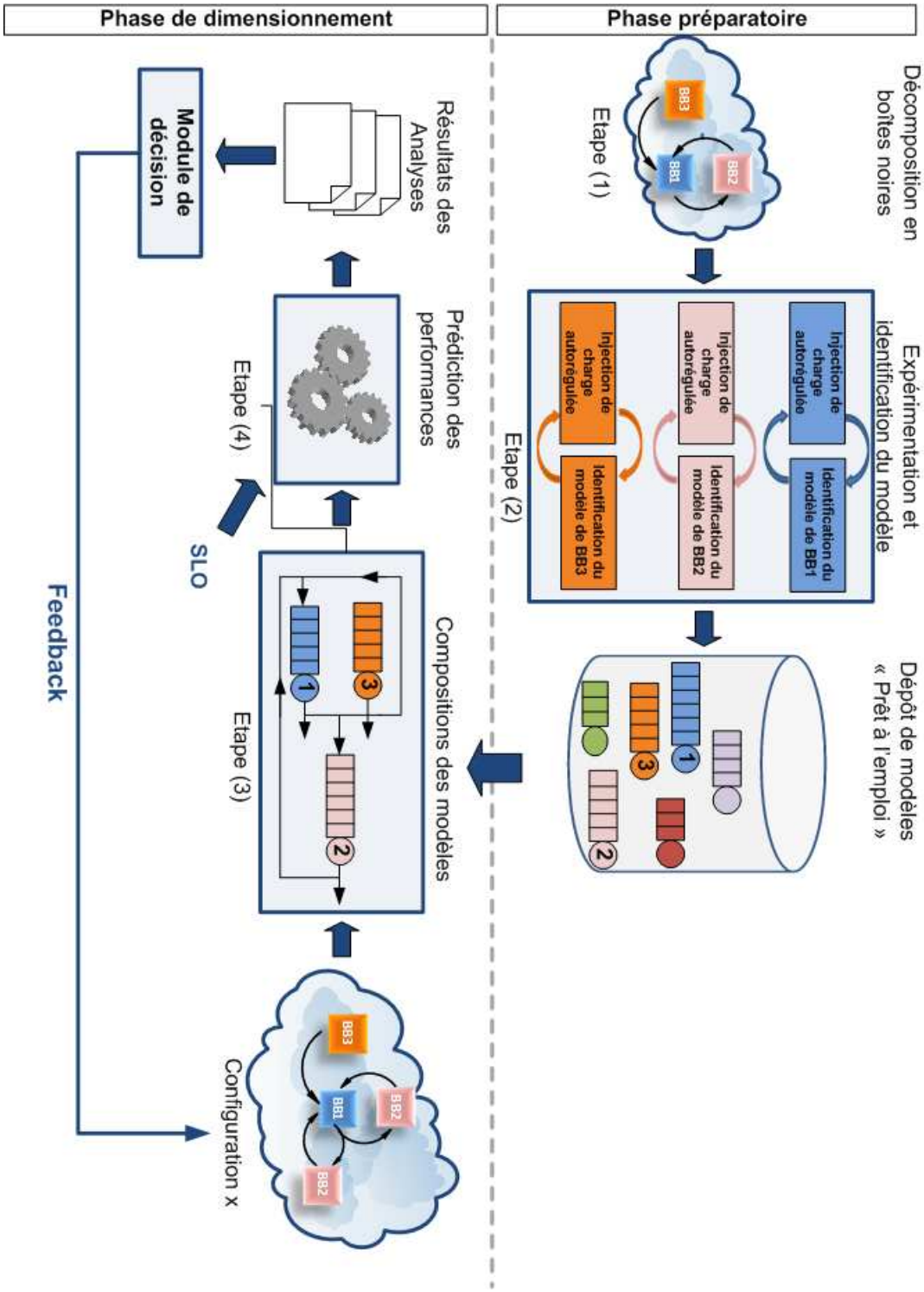

FIGURE 4.1 – Approche globale pour la modélisation automatique et l'aide au dimensionnement

4.2.1 Objectifs

De nombreuses difficultés se posent à l'administrateur d'un système lorsqu'il est confronté à des problèmes de dimensionnement :

- Collecter des mesures fiables sur son système.
- Avoir un modèle de performance aussi détaillé et précis que compréhensible afin d'interpréter le comportement de son système.
- Garantir une bonne qualité de service de son système dans le cadre d'un fonctionnement normal du système.
- ...

Ces objectifs sont souvent très difficiles à atteindre, vue la complexité des systèmes à modéliser et les difficultés des étapes à accomplir pour atteindre ces objectifs.

Le but de notre approche est d'automatiser au maximum les tâches de l'administrateur (l'expérimentation, la collecte des mesures, la modélisation). Elle se propose de :

- Permettre la détection des limites d'un système,
- Fournir une solution de dimensionnement dynamique qui s'adapte aux problèmes imprévisibles liés à l'approvisionnement des ressources,
- Permettre l'extension à une méthode autonomique qui construit des systèmes capables de s'auto-dimensionner. Cette méthode sera également utilisable pour réaliser des diverses propriétés autonomiques : self-sizing, self-optimisation, self-healing,... etc.

4.2.2 Principe

L'idée de base sur laquelle repose notre approche de dimensionnement d'un système consiste à construire d'une manière automatique un modèle de performance du système, puis, en se basant sur la fourniture d'un ensemble de conditions SLOs liés à la qualité de service (QOS) attendue, proposer à l'administrateur des solutions de dimensionnement et une aide à la décision pour réaliser un dimensionnement adéquat.

Nous utilisons à cet effet le formalisme des réseaux de files d'attente pour les raisons citées dans la conclusion du chapitre 1.

Les grandes lignes de notre méthode figurant sur le schéma de la figure 4.1 sont données ci-après :

1. Décomposer le système en plusieurs boîtes noires.
2. Modéliser d'une manière automatique chacune des boîtes noires et fournir un modèle global de la configuration initiale du système.
3. Analyser ou simuler le modèle obtenu et dimensionner le système.

Ces étapes sont détaillées ci-après.

4.2.3 Etapes de l'approche

Notre approche, résumée par la figure 4.1, se constitue de deux phases principales :

a La première Phase : phase préparatoire

La première phase de notre approche est dite **phase préparatoire**. Son rôle essentiel est de préparer des modèles de performance des composants (boîtes noires) d'un système, afin de les utiliser pour l'analyse (d'une manière générale) et le dimensionnement (d'une manière particulière) de diverses configurations d'un système distribué. Pour ce faire, nous procédons en deux principales étapes :

a.1 Décomposition du système : Afin de bien dimensionner un système donné, il est nécessaire de modéliser son comportement d'une manière la plus proche et la plus précise possible. Pour ce faire, nous nous proposons d'abord de décomposer le système en un ensemble de composants vus comme des **boîtes noires**. La décomposition ne peut pas être faite d'une manière automatique puisqu'elle dépend de :

- (i) des choix de l'administrateur (granularité de la décomposition voulue),
- (ii) des questions posées par le problème de dimensionnement,
- (iii) des règles de la décomposition afin de pouvoir caractériser chaque boîte noire.

Les règles de décomposition seront détaillées dans la section suivante.

a.2 Préparation des modèles : La deuxième étape de la première phase consiste à isoler chaque boîte noire, puis identifier automatiquement des modèles de performance associés à chaque type de boîte noire. L'automatisation de cette modélisation est principalement motivée par la complexité qu'un administrateur rencontre lorsqu'il est confronté à un problème de dimensionnement.

En effet, la pratique générale d'un administrateur souhaitant dimensionner un système est de procéder à une batterie de tests à partir desquels des mesures de performance sont récupérées. A partir de ces mesures, est déduit le dimensionnement. Le principal inconvénient de cette méthode manuelle est la perte considérable de temps due à : la complexité de l'expérimentation, la collecte de mesures, l'épuration et la vérification de ces mesures. Vu cet inconvénient, nous avons mis en place un processus d'identification automatique, basé sur une expérience d'injection de charge autorégulée. Ce processus va permettre de récupérer des mesures épurées fiables et d'en déduire automatiquement les modèles de performance associés.

A la fin de cette deuxième étape, un ensemble de modèles est identifié pour chaque boîte noire. Ces modèles sont placés dans un dépôt de modèles. Le dépôt servira principalement à l'analyse d'une configuration du système effectuée dans un objectif donné, cet objectif pouvant être un dimensionnement ou autre (une optimisation, une réparation, etc..).

b La deuxième Phase : Phase de dimensionnement

La deuxième phase de notre approche tente de chercher une ou plusieurs solutions de dimensionnement. Elle part d'une configuration initiale du système à dimensionner et de la définition des SLOs, puis à l'aide des modèles prêts dans un dépôt de modèles pré-construits, elle procède au dimensionnement proprement dit. Pour cela, elle s'opère, comme la phase préparatoire, en deux étapes :

b.1 Construction du modèle global d'une configuration du système : La première étape de la phase de dimensionnement consiste à combiner ou à composer les modèles des boîtes noires constituant le système à étudier suivant une configuration qu'on veut tester. Ces modèles étant définis dans le dépôt. Cette configuration peut être une configuration réelle existante du système qu'on souhaite analyser suite à un problème survenu à notre système, ou bien une configuration de correction ou d'optimisation qu'on veut tester avant de l'appliquer au système en production.

La configuration dans le premier cas correspond à l'interconnexion définie dans la phase de décomposition en boîtes. Dans le second cas, l'architecture peut être une nouvelle proposition de l'administrateur (l'architecte) ou simplement une modification apportée à l'architecture initiale. Le modèle ainsi obtenu par la composition des modèles de boîtes noires, matérialisé par un réseau de files d'attente, représentera le modèle global qui va être étudié dans la prochaine étape.

b.2 Analyse et dimensionnement : L'étape finale est l'étape d'analyse proprement dite du modèle réseau de files d'attente obtenu. Comme chaque boîte noire lui correspond un ensemble de modèles modélisant son comportement, la composition des modèles de boîtes noires donnera lieu à un modèle global pouvant être simulé ou analysé d'une manière exacte. De ce fait découle une précision qui est associée à toute analyse d'une configuration donnée.

Ainsi, cette étape permet, suivant le choix des modèles disponibles dans le dépôt, d'utiliser la méthode d'analyse la plus adéquate au problème traité par l'administrateur. Le choix de la méthode de résolution dépend de plusieurs critères :

– la durée d'exécution que prendra cette méthode,
– la précision associée à l'analyse.

Suite à l'analyse du modèle global, les SLOs (Service Level Objectives) introduits par l'administrateur sont vérifiés. Les résultats de l'analyse et de la satisfaction des SLOs sont fournis à l'administrateur ou à un module de décision pour décider de l'adéquation de la configuration testée à la qualité de service exigée. Dans le cas où cette configuration est adéquate, les modifications seront appliquées au système. Dans le cas contraire, une autre configuration sera analysée à nouveau.

4.3 Problèmes à résoudre

A partir de la description des différentes étapes de notre approche pour le dimensionnement, ressortent plusieurs problèmes d'ordre technique et conceptuel à étudier et à résoudre :

1. Le premier problème est la décomposition d'un système en un ensemble de boîtes noires. Sous quel(s) critère(s) va se faire la décomposition ? Quelle est la granularité de décomposition ? Ces deux facteurs critère et granularité de décomposition influencent énormément sur la résolution du modèle global, ainsi que sur la détection des problèmes liés aux performances.
2. Le deuxième problème auquel nous sommes confronté est l'isolation des boîtes noires. En effet, l'identification du modèle d'une boîte ne doit se faire qu'en se concentrant sur le comportement de la boîte et en abstrayant les autres composants du système. Toutefois, il est nécessaire de prendre en compte l'interaction de cette boîte avec les autres, afin de garantir un modèle correct conforme au comportement de la boîte au sein d'une topologie donnée.
3. Le troisième problème consiste à répondre à la question « comment modéliser ou identifier le modèle d'une boîte noire ». Ceci requiert de réfléchir à une méthode permettant d'englober le comportement global de la boîte pour tout type de charge soumise et de définir un modèle assez précis pour pouvoir comprendre les problèmes et phénomènes surgissant durant la vie du système étudié.
4. Le quatrième problème est la détermination des interactions entre les boîtes noires d'un système. En effet, pour composer les modèles de boîtes noires et construire le modèle global du système, il est indispensable de connaître les communications entre ces boîtes noires et de les quantifier.
5. Le dernier problème, objectif final de notre étude est la réalisation du dimensionnement. Celui-ci doit, non seulement, pouvoir fournir des solutions de dimensionnement à l'administrateur, mais également pouvoir tester ces solutions avant application afin de s'assurer de leur efficacité.

L'ensemble de ces problèmes seront étudiés dans la suite de ce chapitre et les chapitres suivants.

4.4 Décomposition en boîtes noires

4.4.1 Principe

La décomposition d'un système en plusieurs boîtes noires est justifiée par un besoin de finesse du modèle. La finesse du modèle est définie par rapport à la granularité adoptée. Une telle décomposition nous permettra de détecter et d'analyser précisément les problèmes de performance, voire détecter des problèmes non visibles lorsque le système est considéré comme une seule entité monolithique. Pour expliquer cela, prenons l'exemple du problème de détection des goulets d'étranglement. Un modèle qui décrit une application répartie sur plusieurs machines

par un seul modèle de file d'attente ne permettra pas de détecter la machine responsable du goulet. A l'encontre, si on modélise séparément chaque machine par une file d'attente, cela facilitera énormément la tâche de localisation du goulet. Cet exemple nous montre l'utilité de décomposer en plusieurs boîtes noires.

La granularité de la décomposition est un paramètre influent sur la résolution du modèle global, ainsi que sur la détection des problèmes de performances. Plus la décomposition est fine, plus le modèle global du système sous test est précis, plus les problèmes liés à la performance sont mieux appréhendés. Néanmoins, une décomposition fine peut impliquer un modèle global plus complexe : un modèle de file d'attente contenant un grand nombre de stations (de files) prend plus de temps à être analysé ou simulé. Le temps de résolution augmente énormément d'où l'importance de bien choisir la granularité de décomposition.

En plus de la granularité, la complexité et la diversité des systèmes d'aujourd'hui empêchent la définition d'une démarche générique de décomposition. Cette décomposition doit se faire donc d'une façon spécifique à chaque système et elle ne peut donc être automatisée que dans des cas particuliers. Dans le cadre de notre approche, elle se fait d'une manière manuelle par l'administrateur. Différents critères de décomposition peuvent être adoptés.

4.4.2 Types de décomposition

a Décomposition suivant l'architecture logicielle

Le premier type de décomposition est une décomposition suivant le rôle des composants constituant le système sous test. Prenons l'exemple d'une architecture client/serveur. La décomposition suivant l'architecture logicielle consistera à considérer le client comme une boîte noire et le serveur comme une autre boîte, même si cette dernière est répartie sur plusieurs machines (voir figure 4.2).

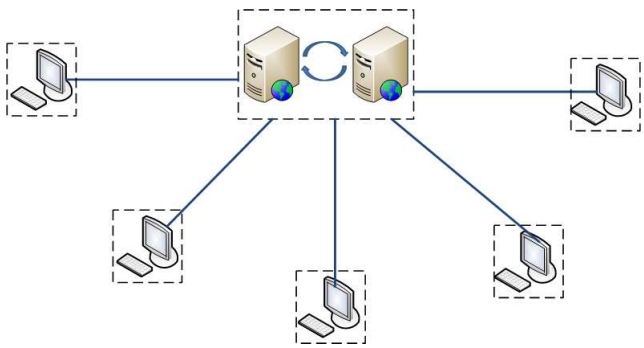

FIGURE 4.2 – Décomposition suivant l'architecture logicielle d'une architecture client/serveur

b Décomposition suivant la localité

Du point de vue pratique, la décomposition la plus simple est celle qui se fait suivant la localité ou suivant l'architecture matérielle. Cette décomposition considère chaque machine physique, quelque soit son rôle dans l'architecture, comme étant une boîte à part. Ce type de décomposition permet de modéliser plus finnement une application répartie que la première décomposition et de détecter précisément la machine responsable du goulets d'étranglement (voir la figure 4.3.a). Toutefois, si on applique cette décomposition à une application largement répartie (voir la figure 4.3.b) le nombre de stations de file d'attente du modèle théorique augmente énormément et le temps de résolution du modèle aussi.

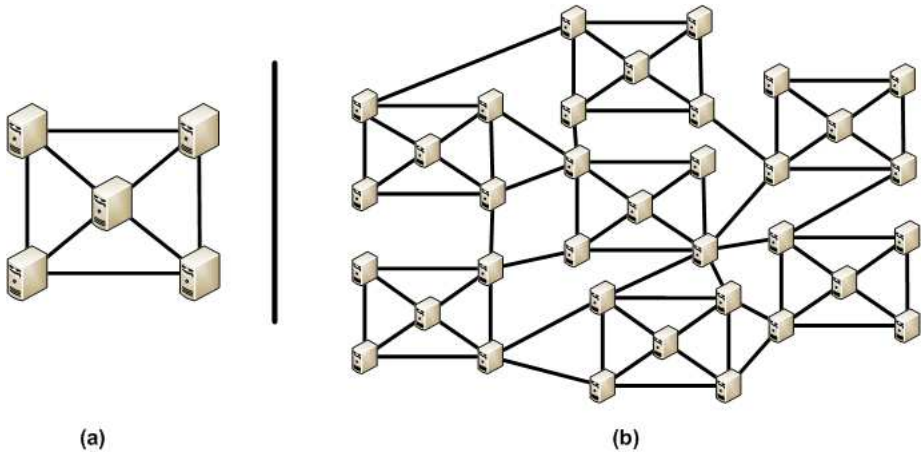

FIGURE 4.3 – Décomposition suivant la localité d'une application répartie

c Décomposition fonctionnelle

Cette décomposition considère chaque unité fonctionnelle applicative comme une boîte noire. Une décomposition fonctionnelle pour une application de commerce électronique considèrera l'unité de gestion des clients comme une boîte noire, l'unité de gestion des paniers comme une deuxième boîte et le catalogue comme une troisième boîte noire. Cette décomposition permet de lier les performances aux fonctions de l'application et de mieux expliquer les éventuels problèmes de performance. Quoiqu'elle nécessite que les différents modules soient suffisamment indépendants pour pouvoir décomposer.

4.4.3 Contraintes techniques et contraintes de connaissances

Il est plus intéressant de choisir une décomposition adaptée aux besoins énoncés en termes d'évaluation de performances et aux moyens d'action. Ainsi, un administrateur qui a comme objectif d'optimiser son application, cherchera à prendre en considération la décomposition fonctionnelle afin d'évaluer ses différents modules fonctionnels. En revanche, il utilisera une

décomposition suivant l'architecture logicielle ou la localité s'il souhaite détecter un goulet d'étranglement matériel. L'administrateur peut aussi combiner les trois types de décomposition pour mieux répondre à ses besoins.

Chaque type de décomposition nécessite un niveau de connaissance différent :

- Une décomposition suivant la localité nécessite simplement la connaissance de l'ensemble des machines sur lesquelles l'application est déployée.
- Une décomposition suivant l'architecture logicielle requiert aussi la connaissance sur la répartition des différents tiers sur l'ensemble des machines.
- Dans le cas d'une décomposition fonctionnelle, l'administrateur doit connaître précisément la répartition des différents modules fonctionnels s'il désire décomposer suivant les unités fonctionnelles.

Etant donné le niveau de complexité des applications réparties étudiées, l'administrateur est souvent amené à décomposer suivant les deux premiers critères (localité et architecture logicielle). Ces deux critères ne posent pas généralement des problèmes liés à la connaissance. En effet, les administrateurs disposent généralement d'informations sur l'architecture matérielle et logicielle nécessaires à une telle décomposition.

Par ailleurs, plusieurs contraintes peuvent limiter une décomposition souhaitée :

- D'une part, la complexité des applications, la complexité des interactions entre ses différents modules ou même l'absence totale de ces informations rendent la tâche de décomposition très complexe, voire impossible.
- D'autre part, une forte dépendance par rapport à un critère de performance peut exister entre les boîtes noires. Ceci incite à ne pas séparer ces boîtes.

Par conséquent, l'administrateur doit choisir un niveau de décomposition permettant de faire plusieurs compromis :

- D'une part, entre le niveau de détail souhaité du modèle et de la capacité à séparer les boîtes (limite technique ou limite de connaissance).
- D'autre part, entre la finesse du modèle et sa complexité d'analyse. En effet, en dépit de l'importance d'avoir un modèle fin afin de bien localiser les éventuels problèmes de performance, cette finesse peut coûter cher en termes de temps d'analyse.

De cette manière, il pourra assurer à la fois un bon niveau de précision du modèle et une durée d'analyse acceptable dans un contexte dynamique et autonomique

4.5 Exemples

Tout au long de ce manuscrit nous allons utiliser un exemple applicatif pour expliquer les différentes étapes de notre approche.

4.5.1 Rubis

Le premier exemple décrit une application Web avec une architecture 3-tiers développée avec différents langages et appelée RUBIS [5]. Rubis est un prototype d'un site d'enchères enligne semblable à *eBay.com*. Le choix d'une telle application est justifié par l'intérêt de comprendre le comportement et d'évaluer les performances de ce type d'application d'enchères en ligne. Ces applications sont assez complexes et attire des millions d'utilisateurs, ce qui leurs confronte souvent à des problèmes liés au dimensionnement. Il est donc important pour l'administrateur de connaître le nombre maximal d'utilisateurs simultanés que le système peut supporter, le(s) composant(s) responsable(s) des goulets d'étranglement. . . La version EJB de Rubis est composée d'un serveur d'application, un conteneur d'EJB et une base de données MySQL (voir figure 4.4).

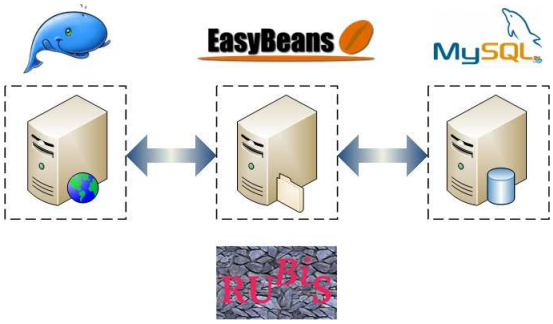

FIGURE 4.4 – L'application Rubis : version EJB

4.5.2 Mystore

Le deuxième exemple est une application e-commerce utilisant le serveur d'application Jonas, cette application nommée *Mystore* est disponible dans le dépôt du projet open source JASMINe [27] du consortium OW2. Contrairement à Rubis, Mystore n'admet pas de tiers base de données.

4.5.3 Sample Cluster

La dernier exemple est l'application *SampleCluster*. Comme son nom l'indique, c'est une application exemple mettant en évidence les fonctions de clustering (création de grappes de serveurs) du serveur JonAS. La figure 4.5 montre l'architecture de *SampleCluster*. La répartition de charge est assurer par le module mod-jk [1] au niveau du serveur apache, et par le protocole d'invocation de méthode à distance en mode cluster(CMI) [2] au niveau EJB.

1. http ://tomcat.apache.org/connectors-doc/
2. http ://jonas.ow2.org/current/doc/doc-en/integrated/Cmi.html

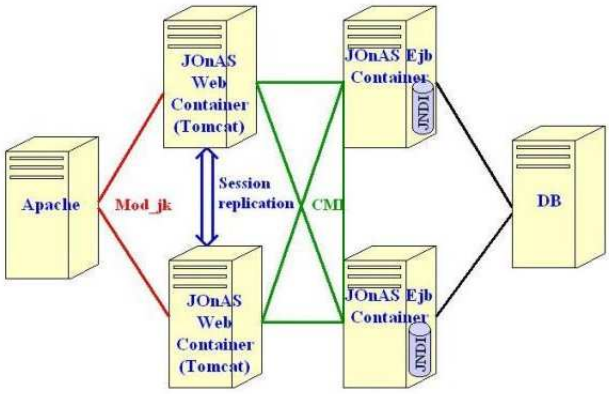

FIGURE 4.5 – Architecture de l'application *SampleCluster*

4.6 Isolation des boîtes noires

Pour des besoins de réparation, d'optimisation ou de reconfiguration, l'administrateur est souvent obligé d'ajouter ou de supprimer une machine, de reconfigurer un ensemble de machines. . . ,etc. Dans ce contexte dynamique de l'architecture et de la configuration des applications, il est recommandé de décomposer le système en plusieurs boîtes avant de le modéliser. En effet, si on modélise tout le système en le considérant comme une seule entité nous serions obligés de ré-identifier ou ré-calibrer le modèle complet après chaque modification. Dans cas de la décomposition, seule le modèle de la boîte sur laquelle porte la modification est identifiée ou ré-calibrer. Surtout lorsqu'on travaille avec des systèmes répartis.

Urgonkar [96] a modélisé chaque tiers d'une architecture 3-tiers par une file d'attente donnant ainsi un modèle de réseau de file d'attente en tandem. Pour cela, il s'est basé sur la mesure du temps de réponse de bout en bout et en estimant le temps de séjour dans chaque tiers par des paramètres internes de chaque boîte. Etant donné qu'on travaille dans un contexte de boîtes noires, on ne pourra pas réutiliser cette méthode qui se base sur des paramètres internes qu'on suppose inconnu d'autant plus lorsqu'on va traiter des réseaux plus complexes que les réseaux en tandem.

Nous avons vu que, pour dimensionner un système, nous procédons d'abord à sa modélisation d'une manière la plus précise possible. La construction d'un modèle global nécessite de modéliser chacune des boîtes noires le constituant. Pour ce faire, nous identifions d'abord le modèle de chaque boîte par un processus d'identification automatique. L'identification du modèle d'une boîte noire doit se faire en se concentrant uniquement sur le comportement de la boîte. Ainsi, elle doit être isolée et expérimentée en isolation.

4.6.1 Motivations

Pourquoi doit-on isoler les boîtes noires ?

Premièrement, les boîtes se trouvant dans une configuration initiale peuvent changer de comportement (donc de modèle), suite à une modification due à une réparation ou une optimisation automatique du système. L'exemple de la figure 4.6 montre que dans la première configuration, la boîte C recevait 70 requêtes par seconde. L'administrateur duplique dans une deuxième configuration la boîte B afin d'augmenter la capacité de traitement du système. Cette modification fait accroître la charge en entrée de la boîte C à 210 requêtes par seconde. Une telle évolution peut soit déclencher un changement du comportement de la boîte de telle manière à s'adapter au nouveau flux ; soit transformer carrément le comportement de la boîte C d'un comportement stable à un comportement non stable. Le modèle doit alors pouvoir représenter le comportement complet de la boîte sous différents niveaux de charge afin d'éviter des situations semblables. La variation de la charge en entrée du système pour observer le comportement total de la boîte peut être limitée par les boîtes en amont. C'est pourquoi il faut pouvoir isoler la boîte et injecter sous différents niveau de charge afin de capturer tout le comportement possible de la boîte.

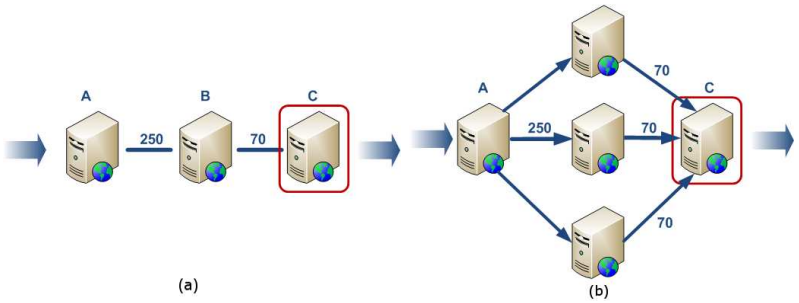

FIGURE 4.6 – Changement du niveau de charge en entrée d'une boîte suite à une modification d'optimisation

Deuxièmement, la boîte peut dépendre des plusieurs autres boîtes qui interagissent avec elle. L'expérimentation de l'ensemble des boîtes vue comme une seule permettra de tester cette boîte sous une seule configuration. Pour les mêmes raisons de dynamicité citées précédemment, il est donc important de tester toute boîte sous différentes configurations. (Cette condition ne sera pas prise en compte dans nos expérimentations néanmoins la méthodologie d'isolation mise en place le permet).

4.6.2 Caractérisation des boîtes noires

Une décomposition souhaitée par l'administrateur peut être non réalisable. Dans ce cas, on considère l'ensemble des boîtes comme une boîte unique. La plupart des systèmes étudiés sont des systèmes répartis sur plusieurs machines et les protocoles de communication utilisés entre les boîtes sont des standards (http, rmi, sql...) qui permettent la capture des interactions en premier lieu, et en second lieu l'injection de requêtes par une injection de charge autorégulée. La caractérisation du comportement complet des la boîte peut se faire de deux manières :

1. En isolant la boîte et en maintenant ou en simulant d'une manière fictive les interactions avec les autres boîtes. Pour cela, nous définissons la notion de bouchon logiciel. Un

bouchon logiciel est une entité logicielle élémentaire, dont le rôle est de remplacer les interactions en amont de la boîte. La figure 4.7 montre la disposition des injecteurs et du bouchon permettant l'isolation et la modélisation d'une boîte noire. Le bouchon est développé de façon à simuler un temps de réponse constant de la boîte. L'objectif est d'extraire le temps de réponse de la boîte, tout en gardant un fonctionnement normal de la boîte afin de mesurer un temps de réponse correcte. Prenons l'exemple de Rubis : si on souhaite caractériser les premiers tiers sans le tiers base de données, nous allons injecter des requêtes http qui interrogeront dans la plupart des cas la base de données. En cas d'absence d'un bouchon SQL remplaçant la base de données, des messages d'erreurs indiquant l'impossibilité d'établir la connexion à la base de données (fonctionnement anormal) remplaceront les réponses HTTP (fonctionnement normal). L'isolation se fait suivant deux phases :

– une première phase où on capture les requêtes entrantes,
– une deuxième phase d'injection de charge basée sur ces requêtes capturées.

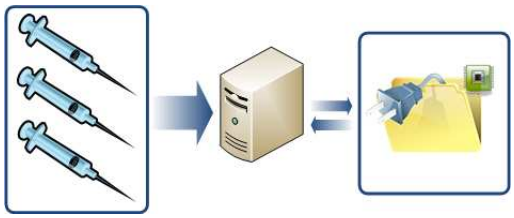

FIGURE 4.7 – Les bouchons logiciels

2. Dans le cas où on aura la possibilité de caractériser le comportement totale d'une boîte sans isolé (Exemple : Cas des boîtes d'entrée de systèmes ou boîtes qui peuvent être saturée), nous utilisons des sondes de captures entre les boîtes afin d'extraire les temps de réponse dans chaque boîte (voir figure 4.8). Cette solution est plus intéressante puisque on n'aura plus le coût de développement des bouchons logiciels. De plus, les boîtes seront expérimentées dans leurs contextes de configuration. Nous utiliserons alors cette méthode comme méthode par défaut pour la caractérisation des boîtes. Seules les boîtes qu'on n'arrive pas à saturer (dont on ne peut pas découvrir le comportement total de la boîte) utiliseront la première solution d'isolation par les bouchons logiciels.

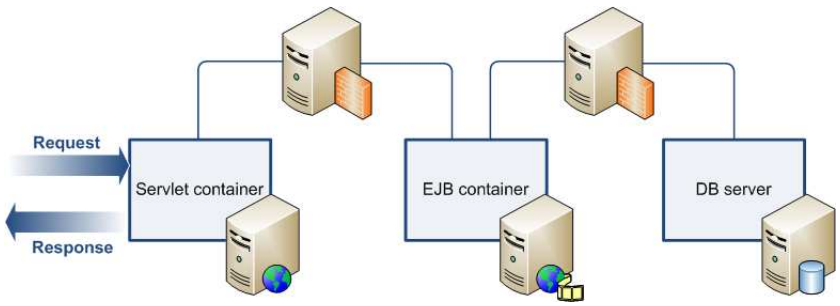

FIGURE 4.8 – Utilisation des sondes pour la caractérisation des boîtes noires

4.7 Conclusion

Ce chapitre a présenté en deux volets une vue d'ensemble résumant notre approche de dimensionnement puis une présentation des problèmes de décomposition et d'isolation. La présentation de ces deux problèmes est importante puisqu'ils montrent notre façon de procéder afin d'obtenir un modèle adapté aux systèmes répartis qu'on traite et au problème de localisation des goulets d'étranglement. D'autant plus que ces deux notions ne seront plus développées dans la suite de cette partie.

Les chapitres suivants détaillerons les différentes étapes de notre approche et se présenterons comme suit : La phase préparatoire qui consiste à expérimenter automatiquement chaque boîte constituant le système afin de générer leur modèle va être développé dans le chapitre 5. Le chapitre 6 montrera comment générer le modèle théorique de chaque boîte en se basant sur les mesures des expériences. Une fois qu'on dispose des modèles de toutes les boîtes qui forment le système, nous montrerons dans le chapitre 7 comment ces boîtes seront composées pour pouvoir construire un modèle global. Enfin le dernier chapitre de cette partie sera consacré à la mise en place de l'environnement de l'expérimentation et de la modélisation automatique implémentant notre approche.

Chapitre 5

Expérimentation automatique pour la modélisation

5.1 Introduction

Ce chapitre a comme objectif de développer et d'expliciter la phase expérimentale permettant la détermination du modèle de chaque boîte noire (voir figure 5.1). On suppose que les boîtes noires sont isolées afin d'effectuer des séries de tests et des expériences permettant de déterminer les mesures nécessaires à l'identification des modèles des boîtes noires. Cette identification sera étudiée dans le chapitre 6.

Dans ce chapitre, deux problèmes fondamentaux seront traités. Le premier est lié à la détermination des mesures qui doivent être à la fois fiables et représentant le comportement complet (comportement à faible, à moyenne et à forte charge) de la boîte sous test. D'où la nécessité de s'assurer, à chaque instant de la collecte des mesures, de la stabilité de notre système. Le second problème est lié aux mécanismes et aux outils nécessaires à l'automatisation de cette phase expérimentale, afin de réduire le temps des expérimentations manuelles, mais sans perdre en qualité d'expérience.

Pour ce faire, nous commençons par présenter le principe d'une expérimentation manuelle, en montrant ses limites. Puis, nous détaillons notre approche automatisant ces expérimentations en vue d'une identification de modèle de performance.

5.2 Expérimentation manuelle de test en charge

Souhaitant collecter les mesures nécessaires à l'analyse des performances d'un système et déterminer la charge maximale supportée, un administrateur effectue plusieurs tests de montée

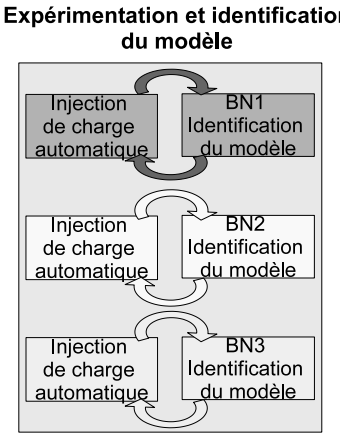

FIGURE 5.1 – Etape 2 : Expérimentation et identification du modèle de chaque boîte noire

en charge. Il s'agit de tests au cours desquels il va simuler un nombre d'utilisateurs sans cesse croissant, de manière à déterminer quelle charge maximale le système est capable de supporter. Il commence alors par injecter une charge minimale et observer les réponses du système sous test. Suivant cette réponse, il décide de la prochaine charge à injecter au système : il augmente la charge lorsqu'il est loin de la saturation du système ; il réduit cette charge lorsqu'il dépasse les capacités du système sous test. Il encadre ainsi la charge maximale supportée.

Cette méthode empirique et manuelle de recherche de la charge maximale est une des activités courantes des experts de benchmarking. Tout au long de ces expériences, l'administrateur collecte les mesures de performances (temps de réponse, consommation mémoire, consommation CPU) induites par chaque charge. Puis, il trace les courbes d'évolution de ces paramètres de performances en fonction de la charge. La figure 5.2 montre l'évolution des temps de réponse moyens d'un système sous différents niveaux de charge. Trois parties peuvent être identifiées sur cette courbe : Une première partie qui exhibe un comportement linéaire des temps de réponses, une deuxième partie qui marque la fin du régime linéaire et où on observe un début de la contention et une dernière partie où on atteint la saturation.

Suivant ses objectifs, l'administrateur cherchera à identifier un point critique appartenant à cette courbe : soit le point qui caractérise la fin du régime linéaire où le paramètre de performance mesuré est proportionnel à la charge injectée, soit un point appartenant à la deuxième partie, qui caractérise une charge maximale supportée lorsque le régime linéaire est dépassé tout en gardant un comportement stable du système. La détermination de la charge de saturation reste une tâche difficile expérimentalement, puisqu'il appartient à la troisième partie où le comportement du système devient instable. On montrera plus tard une estimation théorique de cette charge en utilisant les modèles de file d'attente qu'on aura identifiés.

Remarque : Cette estimation de la charge maximale supportée est très importante dans notre cas, puisque les mesures collectées dans cette phase expérimentale serviront à identifier le modèle de la boîte noire. Dans le cas où on se limite aux mesures collectées à faibles charges, le modèle ainsi identifié ne représentera pas la totalité du comportement du système et le modèle généré manquera par conséquent d'information. D'où la nécessité de connaître également le

comportement du système sous test lorsqu'on frôle la saturation. Toutefois, il faut éviter de dépasser cette charge maximale car les mesures collectées seront instables.

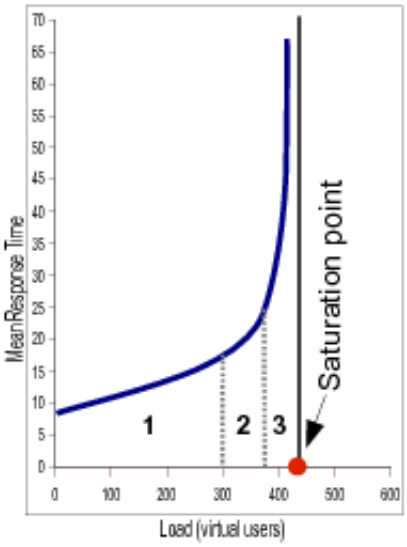

FIGURE 5.2 – Allure de la courbe du temps de réponse moyen en fonction de la charge

Afin de mener une expérimentation de test en charge, l'administrateur se base sur un ensemble d'éléments.

5.2.1 Préparation de l'expérimentation

a Scénario de test en charge

L'injection de charge sur un système sous test se fait en simulant les utilisateurs rééls du système. Le comportement des utilisateurs simulés par les injecteurs de charge doivent se conformer aux comportements des utilisateurs réels. Un comportement est une suite d'actions pouvant être enrichie par des conditions, des branchements, des attentes, des boucles, etc. C'est pour cela que l'administrateur prépare un scénario reprenant les requêtes réelles des utilisateurs et simule des utilisateurs réels, en lançant des utilisateurs **virtuels**.

Reprenons l'exemple de l'application Rubis, un comportement possible d'un utilisateur est la visite de la page de connexion, le choix d'une catégorie d'article, le choix d'un article, l'enchère sur l'article à plusieurs reprises et enfin quitter l'application. L'ensemble de ces comportements représente un comportement d'un seul utilisateur à simuler. On ajoutera des temps d'attente exponentiels qui correspondent au temps de réflexion de l'utilisateur avant de choisir sa prochaine action. Tout ceci constitue le scénario de test en charge à injecter.

Le scénario de test en charge est utilisé pour plusieurs paliers de charge. Les charges testées sur un système constitue ce qui est communément appelé **profil de charge**.

b Profil de charge

On appelle **profil de charge** l'ensemble des charges injectées à un système sous test, au cours du temps de l'expérience. L'administrateur garde une charge constante pendant un temps suffisant pour collecter les mesures de performance associés à chaque niveau de charge (voir figure 5.3). Après avoir analysé ces mesures, il décidera de la prochaine charge à injecter dans sa nouvelle expérience. Il arrêtera ses expériences lorsqu'il détermine la charge maximale que le système peut supporter.

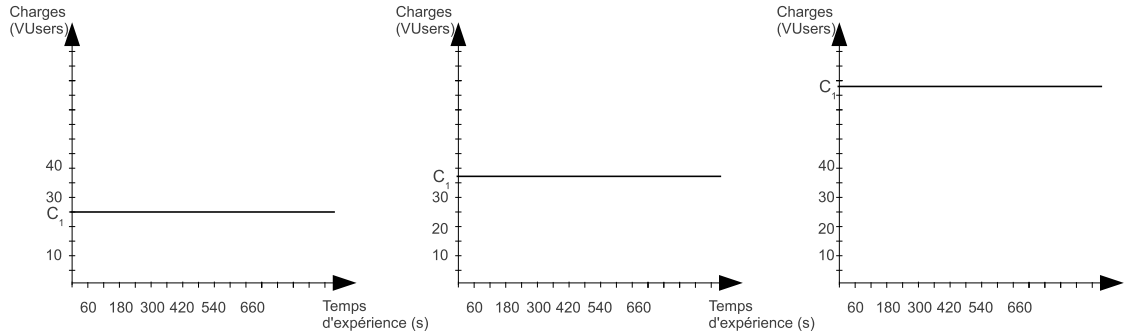

FIGURE 5.3 – Exemple d'expérience avec un niveau de charge fixe

Il faut aussi prendre en considération la répartition de la charge sur plusieurs injecteurs. En effet, cette répartition est utile dans deux cas :

- simuler les aspects répartis des charges émises ou systèmes réels,
- partager la charge à injecter entre plusieurs injecteurs pour ne pas les saturer en cas de charge importante.

5.2.2 Limites de l'approche manuelle

Le principal inconvénient de cette méthode manuelle est la perte considérable de temps, due à la complexité de l'expérimentation, la collecte des mesures, l'épuration et la vérification de ces mesures. A titre d'exemple, la caractérisation des performances d'un serveur d'application J2EE peut prendre une demi-journée d'expérimentation. [1]

De plus, dans un contexte de dimensionnement et de reconfiguration, l'administrateur est amené à refaire ces expériences pour plusieurs boîtes formant le système et pour différentes configurations (partie logicielle, ressources). Nous souhaitons de ce fait automatiser cette méthode de test à travers un contrôle de charge autonome.

1. Estimation approximative donnée par les experts de performance à Orange Labs

5.3 Expérimentation automatique par test de montée en charge autonome

Pour automatiser les expérimentations du test de montée en charge, nous faisons appel aux notions de l'*autonomic computing* et plus précisement à sa boucle de contrôle autonome. L'administrateur sera alors remplacé par un contrôleur autonome capable de :

- analyser les mesures collectées,
- déterminer le profil de charge à injecter au système sous test en fonction des performances observées,
- estimer la charge de saturation et arrêter l'experimentation lorsqu'on s'approche de cette charge,
- déterminer la période de stabilité pour ne collecter que des mesures fiables,
- etc.

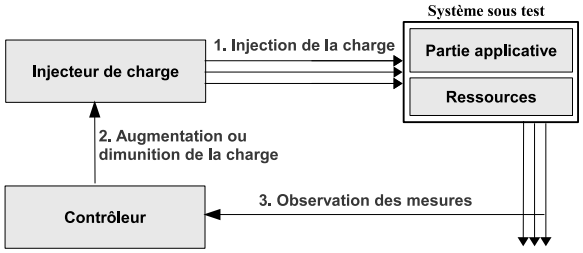

FIGURE 5.4 – Plaforme de test en charge autonome

La figure 5.4 décrit les différentes étapes de la boucle de contrôle pour une expérimentation d'injection de charge autorégulée : Les injecteurs envoient au système sous test des requêtes en gardant une charge constante. Les sondes déployées sur le système sous test collectent les mesures de performances associées. Ces mesures sont envoyées par la suite au composant contrôleur qui analyse ces mesures et décide de la prochaine charge à injecter. La décision dépend d'un modèle intermédiaire identifié par le contrôleur et de la politique d'injection de charge défini par l'administrateur. Le modèle identifié sera utilisé aussi pour estimer la charge maximale.

Les injecteurs reçoivent du contrôleur de la nouvelle charge à injecter et ajoutent le nombre d'ulisiateurs virtuels nécessaires pour assurer le nouveau niveau de charge. L'expérience s'arrête quand le système sous test atteint la zone de saturation. La charge de saturation sera estimée en premier lieu par le modèle théorique intermédiaire identifié, puis corrigée par la suite expérimentalement.

5.3.1 Plates-formes de test en charge

L'automatisation des tests en charge nécessite une plateforme qui soit capable d'enchaîner des actions sur le système sous test selon des scénario. Plus particulièrement, elle doit pouvoir disposer de :

- un mécanisme de création d'*utilisateurs virtuels* et de construction d'un scénario de test en charge qui va être exécuté par ces utilisateurs.
- un mécanisme d'injection des actions composant un scénario de test en charge. Par exemple, pour tester un serveur d'application web, il est nécessaire de pouvoir injecter des actions http pouvent être des requêtes Post, Get, etc.. . Pour tester un serveur de base de données, les actions à injecter sont des requêtes SQL Insert, Select, Update, ...
- des injecteurs de charge qui invoquent le système sous test. Ils envoient les requêtes, simulent le comportement des utilisateurs, attendent les réponses et mesurent un ensemble d'indicateurs de performances tels que le temps de réponse du système.
- des sondes réparties qui mesurent la consommation des ressources du système sous test ainsi que les ressources des machines sur lesquelles elles sont déployées. Plusieurs métriques de ressources sont mesurables : par exemple la consommation CPU, l'utilisation mémoire, l'utilisation de la mémoire de la JVM, la consommation disque, le traffic réseau (paquets réseaux émis et reçus), etc.

CLIF [2] est une plate-forme d'injection de charge conçue selon le modèle à composants modulaire et extensible Fractal [3]. Il peut être utilisé pour injecter plusieurs types de systèmes cibles tels que les serveurs HTTP, les bases de données, les serveurs DNS, les serveurs SIP, etc. CLIF permet de déployer sur un réseau et d'administrer à distance des sondes et des injecteurs (voir figure 5.5). Ces derniers permettent d'injecter une charge de requêtes réparties sur le système sous test, d'attendre les réponses et de mesurer les temps de réponse moyens d'une requête. Les sondes mesurent la consommation des ressources des machines sur lesquelles elles sont déployées (Système sous test, Serveur d'injection, etc). Plusieurs types de ressources sont mesurables par les sondes CLIF : la consommation CPU, l'utilisation mémoire, l'utilisation de la JVM, la consommation disque, etc.

Parmi les points forts de la plate-forme CLIF est l'utilisation des scénarios ISAC permettant la définition et la gestion de la charge à injecter. Plusieurs comportements peuvent être spécifiés aux groupes d'utilisateurs virtuels formant la charge. Les comportements peuvent être réalistes moyennant les structures d'instructions et de contrôles fournies (boucles, tests, attentes . . .). La population de chaque groupe suit un profil de charge qui lui est propre. Le composant superviseur permet de gérer et d'administrer l'injection de charge.

Le choix de CLIF est alors justifié par sa conformité aux contraintes citées précédemment et de sa flexibilité et extensibilité grâce à son système de plugins.

5.3.2 Profil de charge automatique

Le protocole expérimental manuel est itératif, à partir de l'observation faite pour un niveau de charge donné on calcul le niveau d'injection de charge suivant ainsi que les paramètres associés à l'observation suivante. l'objectif est alors d'automatiser ce protocole, de calculer, en ligne

2. http ://clif.ow2.org/
3. http ://Fractal.ow2.org/

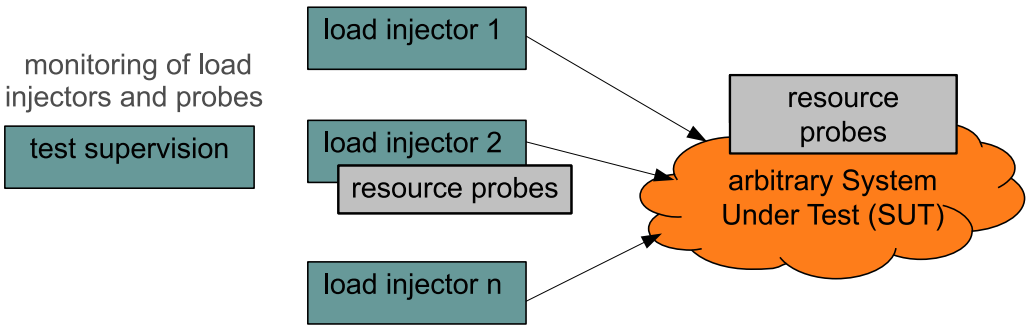

FIGURE 5.5 – Architecture de la plate-forme CLIF

les différents paramètres de l'observation à un niveau de charge et de déterminer et transmettre ces paramètres au contrôle d'injecteur pour préparer l'injection suivante. Cette méthode permet en contrôlant dynamiquement la montée en charge, d'éviter l'initialisation du système (cas de l'approche manuelle) et de maîtriser la variabilité du système liée au changement de la charge injectée aux différents niveaux.

Pour automatiser ces expériences, nous avons choisi une solution qui réduit le nombre d'expériences de l'approche manuelle à une seule dont le profil de charge est une fonction par palier (ou par escalier). Chaque palier correspond à une expérience manuelle. Les paramètres du i^{eme} palier sont fonctions des paramètres de performances mesurés au palier précédent et de plusieurs autres éléments.

Ces éléments sont déterminés à deux niveaux. Le premier niveau est le niveau des paramètres initiaux défini par l'expérimentateur :

– la précision des estimateurs utilisés,
– la politique d'injection de charge,

Le second est le niveau des fonctions de calcul ou des estimateurs. Ces fonctions décrivent les modèles et les choix prédéfinis

– la charge maximale estimée C_{max},
– la durée d mesurée dans le palier précédent,
– le temps de stabilisations.

Le niveau de précision des estimateurs est défini à la phase de préparation de l'expérience. Plusieurs autres paramètres sont déterminés au vue des mesures de palier précédent ce qui signifie que plusieurs paramètres seront déterminés à la volet.

la figure 5.6 décrit un profil de charge automatique et ses différents paramètres à déterminer.

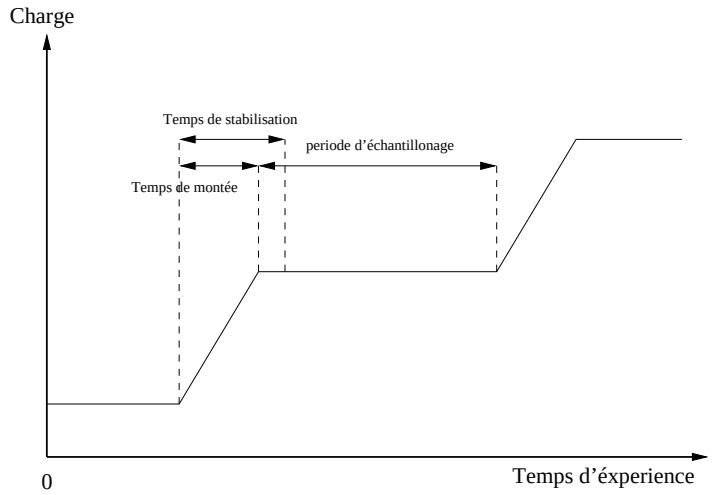

FIGURE 5.6 – Exemple de profil de charge en palier

5.3.3 Contraintes d'automatisation du profil de charge

Contrairement à la méthode manuelle, l'expérimentation automatique ne pourra pas injecter une charge supérieure à la charge maximale. En effet, le dépassement de la charge maximale peut laisser le système sous test dans un état instable même après la diminution de la charge. Dans le cas manuel, l'administrateur arrête le système sous test et recommence l'expérience en prenant en considération cette nouvelle donnée. En revanche, dans un processus d'expérience automatique, cela nous amène à arrêter l'ensemble des expériences et à reprendre le test en charge dès le début. Cependant, il est impératif pour les systèmes où l'état d'instabilité est irréversible d'approcher la charge de saturation par valeur inférieure. D'où, l'importance d'estimer la charge maximale supportée par le système dès le début de l'expérimentation.

Le choix de la pente entre deux paliers du profil de charge influe la durée de la montée en charge et le temps total de l'expérimentation. Mais le plus important est son influence sur le temps de stabilisation. Plus la pente est verticale plus le risque de perdre la stabilité du système est plus grande. Et cela surtout lorsqu'on est proche de la zone de saturation ou lorsqu'on a un grand incrément de charge entre deux paliers successifs. Par conséquent, il est important de choisir une pente qui réduit les risques d'instabilité sans autant trop augmenter le temps global d'expérimentation (voir section 5.3.4).

5.3.4 Paramètrage de l'expérimentation

a Politique d'injection

La politique d'injection de charge est l'algorithme défini par l'expérimentateur et qui permet de décider du pas d'injection (ou l'incrément). Ce pas doit être choisi soigneusement (prudemment). En effet, un grand incrément réduira le temps global de l'expérimentation (moins de paliers) mais augmentera le risque de perdre la stabilité du système en dépassant la charge maximale supportée par le système sous test. Un petit pas permet d'approcher au mieux la charge maximale mais entraîne une perte inutile du temps d'expérimentation dans les zones linaires (voir figure 5.2). L'expérimentateur doit alors exprimer sa politique d'injection en fonction des indices de performance du système. Parmi ces indices, on peut utiliser le temps de réponse, la charge CPU, ou plus simplement une estimation de la charge maximale supportée. Cette estimation doit prendre en considération tout indice de performances influant la saturation du système (voir section 5.3.5). Une fois cette charge maximale estimée, l'expérimentateur choisit un algorithme qui permet de rapprocher cette charge estimée en tenant compte de la rapidité de l'expérimentation, la stabilité du système et du nombre de paliers minimal et nécessaire pour l'estimation du modèle du système sous test.

La politique d'injection peut être un rapprochement par dichotomie, ou un incrément inversement proportionnel à la distance entre la charge actuelle et la charge maximale.

Exemple d'algorithme :

dichotomique :

$C_{suivant} = \frac{C_{max} - C_{prcdent}}{2}$

Autre algorithme :

si $C_{prcdent} < C_{max}/2$ Alors $C_{suivant} = C_{prcdent} + pas$
sinon $C_{suivant} = C_{prcdent} + \frac{pas}{2}$

b Temps de montée en charge d'un palier

Pour éviter de déstabiliser le système sous test dans les phases transitoires entre deux paliers successifs, il est important d'ajouter progressivement la charge. Ceci permettra d'empêcher un probable dysfonctionnement dans le cas d'un grand incrément et par conséquent l'arrêt de l'expérimentation automatique.

Dans la pratique, l'expérimentateur peut fixer un temps de montée en charge ou définir un

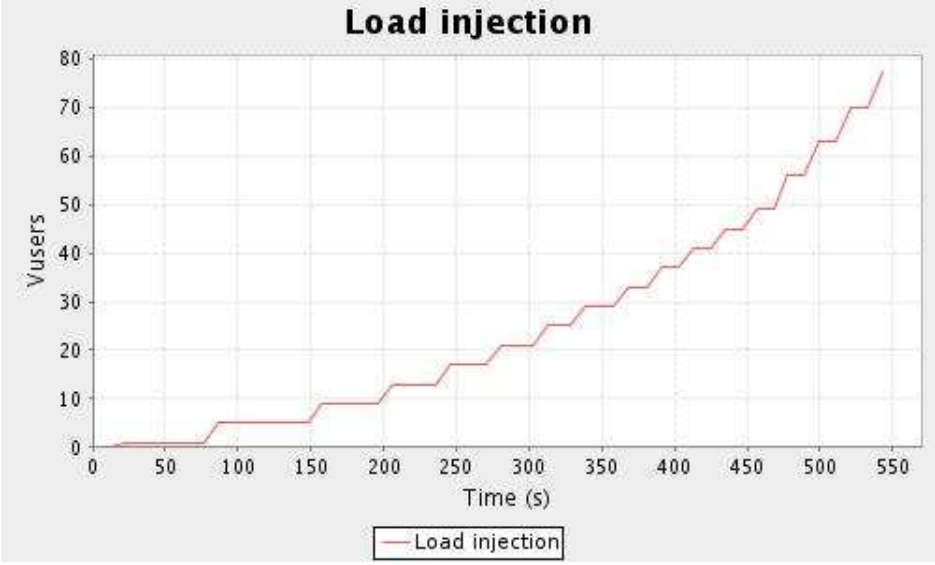

FIGURE 5.7 – Montée en charge de l'application Mystore

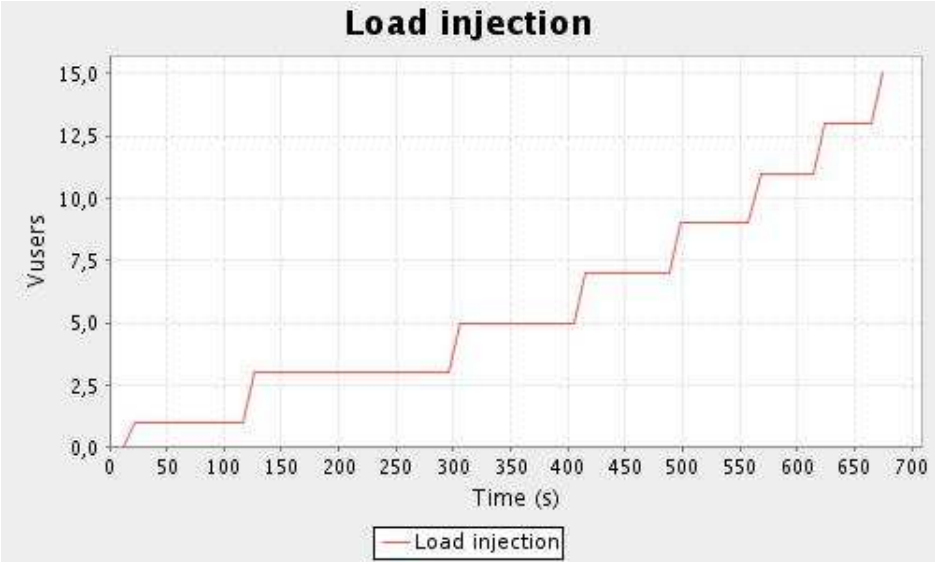

FIGURE 5.8 – Montée en charge de l'application Sample Cluster

seuil sur le taux d'injection (exemple : seuil d'injection = 10 Vusers/seconde)

c Temps d'injection

La durée de la période d'injection doit être suffisamment longue afin d'avoir des mesures fiables et précises. D'une part, cette période doit être supérieure au temps de stabilisation estimé. Et d'autre part, l'échantillon doit être suffisamment grand pour avoir une bonne confiance dans les mesures. La fonction qui lie la précision à la période d'injection dans un palier est décrite

dans la section 5.3.5. Toutefois, l'expérimentateur doit décider de la précision (en pourcentage) ou de l'intervalle de confiance des mesures nécessaires pour la détermination de la taille de l'échantillon et ensuite la durée échantillonnage.

5.3.5 Fonctions

a Fonction d'estimation de la charge maximale de fonctionnement initial

La définition de la montée en charge nécessite l'estimation de la charge maximale de fonctionnement afin d'assurer la stabilité du système sous test, et ça, dès les premiers paliers. Cette charge étant proche de la charge de saturation ce qui rend la détermination de sa valeur très difficile expérimentalement. L'idée est alors d'estimer cette valeur en partant d'un modèle général qu'on améliorera au fur et à mesure de l'avancement de l'expérience (voir chapitre 6). Le modèle choisi pour chaque boîte noire sous test étant une file d'attente (voir la figure 3.2). Nous cherchons alors à estimer cette charge maximale en se basant sur le modèle général G/G/k.

En absence d'un ordre de grandeur de la valeur $\hat{C_{max0}}$, On commence à injecter une seule requête à notre boîte noire. Cette démarche permet non seulement d'avoir une charge minimale acceptable mais d'avoir des résultats intéréssants pour notre estimateur $\hat{C_{max0}}$.

Notons $\overline{R}$ le temps de réponse moyen et $\overline{W}$ la moyenne du temps d'attente.

Nous avons : $\overline{R} = \overline{W} + \overline{X}$.

Lorsqu'on reduit le nombre des requêtes (clients) à un seul (pas de concurrence), $\overline{W} = 0$, ce que signifie que :

$$\overline{R} = \overline{X} = \frac{1}{\mu}$$

On note ce temps de réponse moyen par $\overline{X_0}$ et le taux de service correspondant par μ_0. La fonction du temps de réponse moyen en fonction de la charge du modèle G/G/k admet une asymptote verticale en $k.\mu$(voir la figure 5.9). En supposant que l'unique requête injectée initialement sera traitée par un seul serveur, on obtient une première approximation de $\hat{C_{max0}} = \mu_0$. Cette valeur de $\hat{C_{max}}$ sera par la suite corrigée expérimentalement en fonction du nombre de serveurs identifiés (voir section 5.3.6).

b Fonction du calcul du durée de l'échantillion

Selon [44], la détermination de la taille d'un échantillon nécessaire pour obtenir un niveau de précision souhaité, est défini par inégalité :

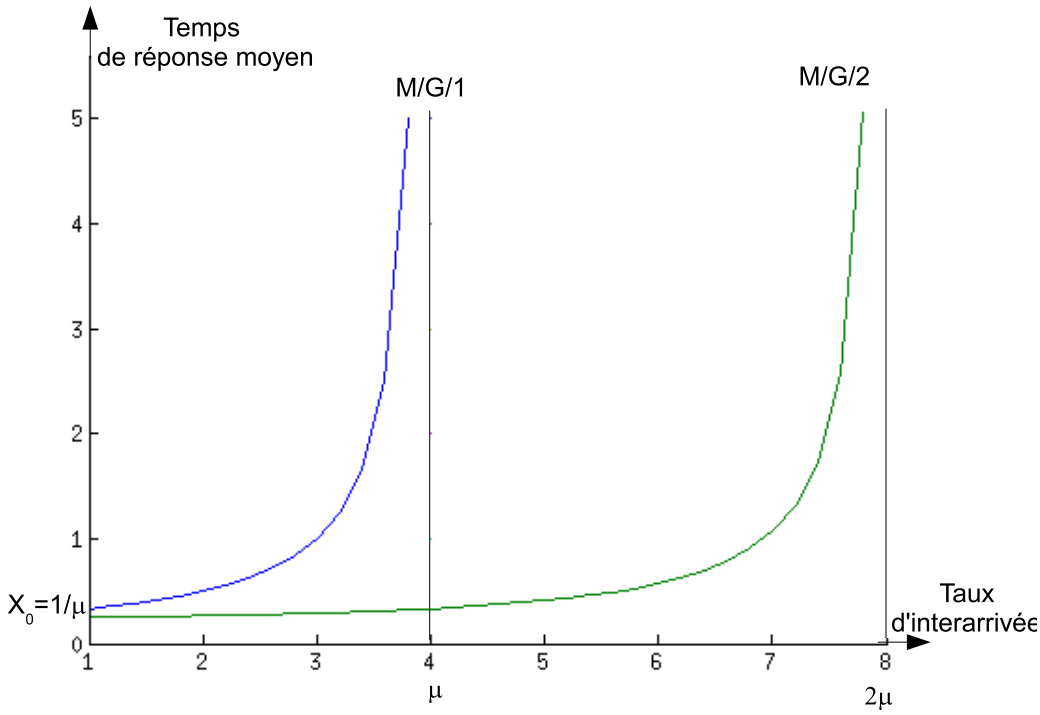

FIGURE 5.9 – Temps de réponse moyen pour la G/G/k (pour k=1 et k=2)

$$n \geq (\frac{100 z \sigma_n}{r \bar{m}})^2$$

avec
- r : le niveau de précision,
- n : la taille de l'échantillon,
- $ic = 100 * (1 - \alpha)\%$: l'intervalle de confiance souhaité,
- z : la variable normale standard associée au niveau de confiance,
- $\bar{m}$: la valeur moyenne du paramètre à estimer,
- σ_n : l'écart type d'échantillon.

c Estimation du temps de stabilisation

Lors de la collecte des mesures, il est important de distinguer la période transitoire. Une première idée serait de se baser sur la variance des mesures pour décider que l'on a atteind le régime stationnaire. Malheureusement, cette méthode n'est pas fiable. En effet, la présence de pics dans les mesures expliqués pour partie par des effets du *garbage-collector* rend cette méthode inexploitable. Par conséquent, une approche plus fine, basée sur une étude théorique et experimentale est nécessaire.

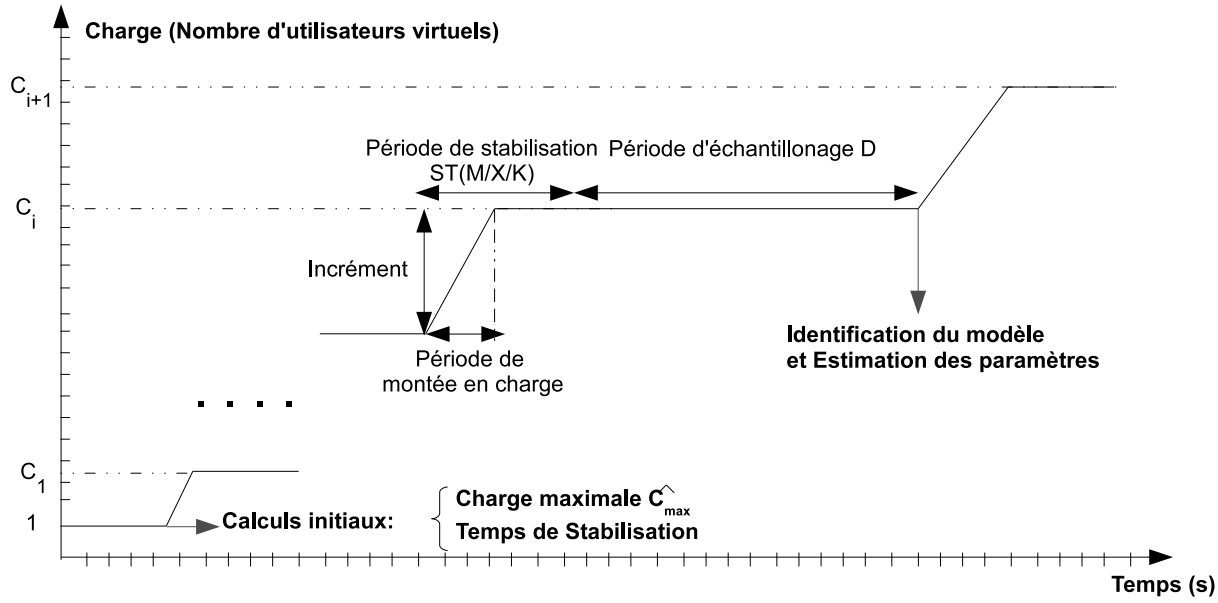

FIGURE 5.10 – Calcul de la longueur du palier nécéssaire

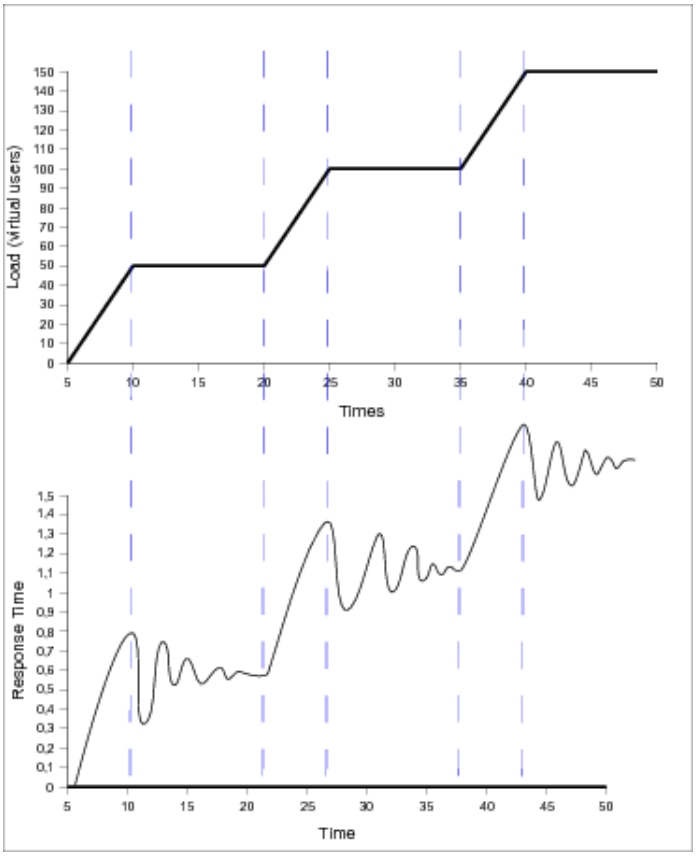

FIGURE 5.11 – Calcul du temps de la montée et du temps de stabilisation

A chaque étape (i), on calcule un temps de stabilisation théorique, défini comme le temps de

convergence de la chaîne de Markov sous-jacente [50, 91] associée au modèle de file d'attente de l'étape.

Le temps de stabilisation, noté ST, est calculé à partir de l'étude du comportement transitoire du système. Comme le modèle de file d'attente de l'étape (i) n'est pas encore défini, nous nous appuyons sur le modèle de file d'attente de l'étape précédente (i-1). On suppose que la variation du temps de stabilisation de la charge ne varie pas trop lorsque l'incrément de la charge entre deux étapes successives n'est pas importante. Au pire cas, cette variation sera capturée par la partie expérimentale.

Soit P la matrice de transition du modèle de file d'attente de l'étape (i-1). P est une matrice CxC avec C une valeur supérieure au nombre des requêtes attendues dans le système. Cette valeur doit être choisie telque la probabilité que le nombre des requêtes se trouvant dans une file M/M/1 équivalente dépasse C est inferieur à ϵ tres petit.

$$P_{M/M/1}(N \geq C) \leq \epsilon$$

L'étude du comportement transitoire consiste à déterminer le vecteur probabilité $\pi^{(n)}$ au moment n, tel que :

$$\pi^{(n)} = \pi^{(n-1)}P = \pi^{(0)P^n} \ldots (1)$$

Le vecteur de probabilité à état d'équilibre π est alors défini comme :

$$\pi = \lim_{n\to\infty} \pi^{(n)} = \lim_{n\to\infty} \pi^{(n-1)}P, \text{ and so } \pi = \pi P$$

En supposant que la limite existe, on calcule le temps de stabilisation ST en fonction du nombre n d'itérations nécessaires pour atteindre l'état d'équilibre. Donc, pour obtenir n, on résout itérativement l'équation (1).

Une des méthodes numériques utilisées pour résoudre l'équation $\pi = \pi P$ est la *méthode de puissance* [91]. Cette méthode est bien connue dans le contexte de la détermination du vecteur propre correspondant aux valeurs propres dominantes de la matrice P. Elle est décrite par la procédure itérative suivante :

$\pi^{(n+1)} = \frac{1}{\xi_n} P\pi^{(n)}$, avec ξ_n est un facteur de normalisation et $\pi^{(0)}$ est le vecteur de probabilité initial.

D'après Stewart [91], si P a n vecteurs propres $(v_i)_{i=1,\ldots,n}$ associés aux n valeurs propres $(\alpha_i)_{i=1,\ldots,n}$ tel que $|\alpha_1| > |\alpha_2| \geq \ldots \geq |\alpha_n|$, et le vecteur initial peut être écrit sous forme d'une combinaison linéaire des vecteurs propres de P, $\pi^{(0)} = \sum_{i=1}^{n} a_i v_i$, le taux de convergence est alors déterminé à partir de la relation :

$$\pi^{(n)} = \pi^{(0)} P^n = \sum_{i=1}^{n} a_i \alpha_i^k v_i = \alpha_1^k \left\{ a_1 v_1 + \sum_{i=2}^{n} a_i \left(\frac{\alpha_i}{\alpha_1} \right)^k v_i \right\}.$$

Dans ce cas, la vitesse de convergence dépend du rapport $\frac{|\alpha_i|}{|\alpha_1|}$ pour $i = 2, \ldots, n$. Plus ces rapports sont petits, plus la somme tend plus vite vers zéro. En particulier, comme P est une matrice stochastique, la première valeur propre α_1 est égal à 1, et ainsi la valeur propre dominante α_2 détermine le taux de convergence.

Par conséquent, la rapidité avec laquelle les quantités $\pi^{(n)}$ convergent dépend des valeurs propres de P [91] et seulement des 2 premières valeurs propres pour le cas particulier de M/M/1/C.

Dans notre cas, nous calculons théoriquement le temps de stabilisation en déterminant la première matrice P à partir du générateur infinitésimal tridiagonale tel que :

$$P = Q\Delta t + I, \text{ avec } \Delta t \leq 1/max_i|q_{ii}|.$$

Nous choisissons une valeur de $\Delta t = \frac{1}{\lambda+\mu}$, où λ est le taux d'arrivée et de μ est le taux de service déduit à partir du temps de réponse moyen obtenu à l'étape(i-1).

On résout itérativement l'équation(1) et on s'arrête une fois la différence

$$||\pi^{(n)} - \pi^{(n-1)}|| < \epsilon$$

, avec ϵ fixé.

Le temps de stabilisation est alors : $ST = \frac{n}{\lambda+\mu}$, λ le taux des inter-arrivées de l'étape (i) et μ est approché par le taux de service du modèle de l'étape précédente $model_{(i-1)}$.

Comme on se base sur $model_(i-1)$, ce qui n'est pas nécessairement (proche) du modèle de l'étape (i), on corrige le temps de stabilisation obtenu en déduisant une erreur ϵ_i, calculée expérimentalement en observant le coefficient de variation des mesures collectées à l'étape (i).

d Évolution du temps de stabilisation

Avant de se lancer dans l'estimation du temps de stabilisation théorique, il est intéressant d'étudier son évolution en fonction de la charge dans le cas général. Étant donnée la formule $ST = N_t/(\lambda + \mu)$, l'évolution de ST dépend linéairement de N_t. Cette valeur représente le nombre de transitions nécessaires pour arriver à état stationnaire à chaque palier pour la chaîne de Markov uniformisée. Dans le cas d'un grand incrément de la charge, le système mettra plus de temps à retrouver l'état stationnaire (N_t grand) et cela risque de se produire dans les premiers paliers. Contrairement aux étapes finales, lorsqu'on est proche de la saturation, les politiques d'injection prévoient généralement de diminuer l'incrément afin d'éviter de dépasser le C_{max}

estimé. Ce qui signifie que le comportement du système reste proche du comportement stationnaire, induisant un retour rapide au régime stationnaire (N_t petit). De plus, le dénominateur croît en fonction de la charge rendant le temps de stabilisation plus petit lorsque la charge augmente.

En résumé, l'évolution du temps de stabilisation dépend des trois variables qui évoluent de façons différentes. D'une part, l'état initial et l'inverse des débits des événements $\frac{1}{\lambda+\mu}$ qui selon notre expérimentation décroisent. Et d'autre part, le nombre d'itérations qui augmente lorsqu'on s'approche du point de saturation. Les expérimentations que nous avons réalisés montrent que les deux premières variables dominent et font décroître le temps de stabilisation (voir le tableau 5.1). Selon [58] pour le cas des files d'attente de type M/M/1/C, le temps de relaxation qui est défini comme le temps nécessaire pour atteindre le régime stationnaire (correspondant à notre temps de stabilisation) augmente en fonction du nombre de client se trouvant dans la file. Cette différence peut être expliquée en partie par la manière dont nos politiques d'injection sont définies mais aussi à d'autres facteurs liés au système sous test qui seront exposés dans la section suivante.

e Exemple

Reprenons maintenant l'exemple de Mystore. Le temps de stabilisation théorique estimé décroît rapidement lorsque la charge augmente. Ce qui semble conforme à nos prévisions. Mais cette décroissance est plus rapide que prévue puisque l'on obtient un temps de stabilisation proche de zéro alors qu'on est dans une zone instable s'approchant de plus en plus d'un point de saturation. Il est alors important de prendre en considération la distance de la charge par rapport à l'état de saturation et d'utiliser une borne maximale à ce temps.

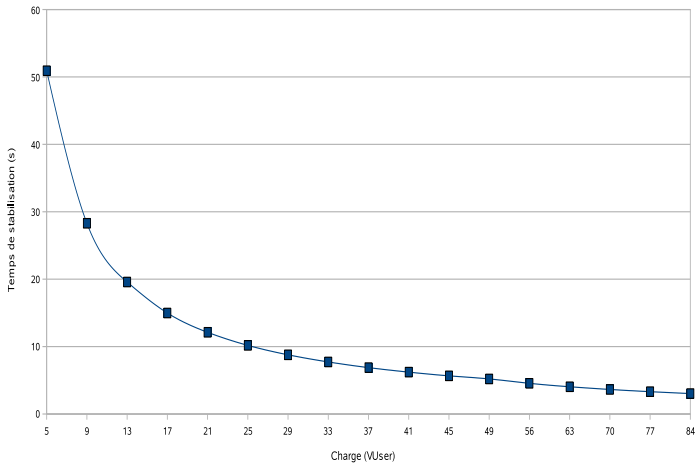

FIGURE 5.12 – Evolution des temps de stabilisation en fonction de la charge injectée

Pour l'exemple de Rubis, la décroissance est moins rapide et la valeur estimée peut être utilisée jusqu'à la fin de l'expérimentation. Cependant l'existence des valeurs extrêmement grandes au niveau des premiers paliers peut être expliquée par l'effet de démarrage de l'application qui

Palier	Charge injectée	Temps de stabilisation théorique
1	1	-
2	5	50.92
3	9	28.29
4	13	19.58
5	17	14.98
6	21	12.12
7	25	10.18
8	29	8.78
9	33	7.72
10	37	6.88
11	41	6.21
12	45	5.66
13	49	5.20
14	56	4.55
15	63	4.04
16	70	3.64
17	77	3.31
18	84	3.03

TABLE 5.1 – Mystore : temps de stabilisation en fonction de la charge injectée

nécessite dans plusieurs cas une étape de chargement de module ou par plusieurs passage du *garbage collector*.

Palier	Charge injectée	Temps de stabilisation théorique	Commentaires
1	1	341.66	valeur non significative
2	20	88310	phase de démarrage et de chargement
3	50	9758	phase de démarrage ou passage du gc
4	70	1871	décroissance normale
5	100	614	
6	120	425	
7	150	419	
8	200	323	

TABLE 5.2 – Rubis : temps de stabilisation en fonction de la charge injectée

5.3.6 Correction de la charge maximale de fonctionnement et identification du nombre de serveurs

Nous avons montré dans la section 5.3.5 que la politique d'injection doit être en fonction de l'estimation de la charge maximale. Toute la réussite de l'expérimentation dépend de la bonne estimation de cette charge maximale. D'où l'importance de corriger cette valeur au fur et à mesure de l'avancement de cette expérimentation. La figure 5.9 montre que le temps de réponse moyen du modèle général GI/GI/k admet une assymptote verticale à chaque $k\mu$. Cette valeur peut être considérée comme une estimation de la charge maximale en fonction du nombre de serveurs. Par conséquent, l'identification du nombre de serveurs k permettra de corriger à la volée notre estimateur de charge maximale.

La figure 5.13 montre comment déterminer le nombre de serveur en fonction de la charge actuelle, la charge maximale estimée et de l'état du système (saturé ou non). En effet, cette méthode itérative consiste à comparer la charge en cours en fonction de la charge maximale, si on observe qu'on est proche de cette charge alors que le système est loin de son état de saturation (taux utilisation faible), on incrémente k. L'observation d'une forte utilisation des ressources du système sous test arrêtera l'incrémentation de k et estimera de cette manière le nombre de serveur associés au modèle à identifier.

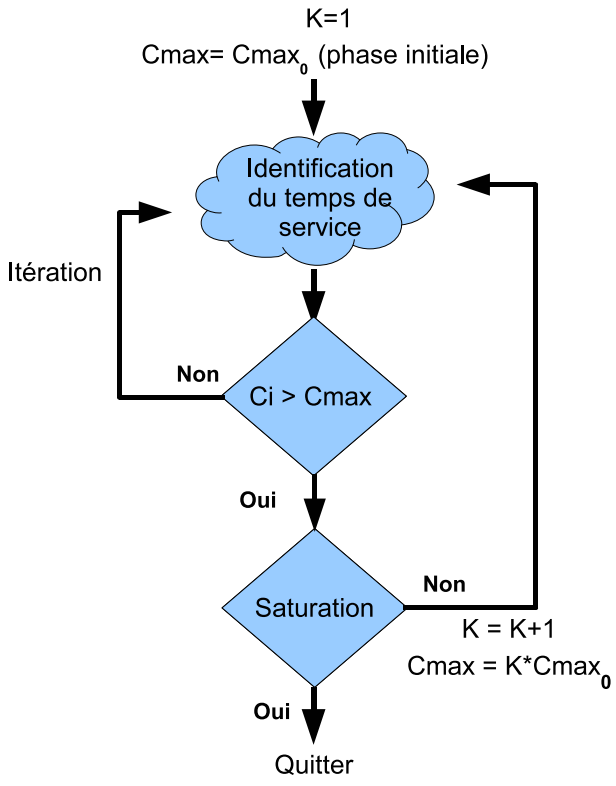

FIGURE 5.13 – Détermination du nombre de serveurs d'une boîte noire

5.4 Choix des mesures collectées

La détermination des paramètres (C_{max},k, ST ...) s'appuie sur un modèle identifié à chaque étape de l'injection. Ce modèle se base sur les mesures collectées dans les étapes précédentes de l'expérimentation. Il est donc impératif de choisir les mesures suivant deux critères :

- des résultats théoriques à disposition permettant l'identification et le calibrage du modèle ;
- des ressources surveillées afin d'estimer le taux d'utilisation de notre système et détecter la zone de saturation.

Pour la majorité des cas étudiés, nous avons eu besoin :

- Des mesures du temps de réponse et des instants d'arrivée des requêtes sur le système sous test permettant d'extraire les temps de service (voir chapitre 6) ;
- du taux d'utilisation des ressources : la CPU, la mémoire et la JVM, permettant d'observer l'utilisation des ressources et de déterminer un estimateur de l'utilisation ;
- Le débit pour le calibrage et la validation du modèle obtenu.

5.5 Conclusion

Ce chapitre a montré les mécanismes nécessaires à l'automatisation de l'expérimentation d'injection de charge. Après avoir exposé les contraintes d'automatisation, nous avons présenté et justifié le choix du profil de charge utilisée. Ensuite, nous avons exposé les différents paramètres de l'injection définie par l'expérimentateur et les fonctions qui complètent les aspects restant du profil de charge. Ces fonctions se basent sur des choix et des modèles afin d'assurer un bon déroulement de l'expérimentation et des mesures fiables.

La dernière partie de ce chapitre a été consacrée au choix des mesures à collecter et qui seront fournie à la phase d'identification de modèle de chaque palier qui sera traité dans le chapitre suivant.

Chapitre 6

Identification d'un modèle et rétroaction

« *Measurements are not to provide numbers but insight.* » Ingrid Bucher

6.1 Introduction

La phase expérimentale expliquée dans le chapitre précédent se fait en plusieurs étapes. Chaque étape consiste à injecter une charge donnée et à fournir une partie des mesures nécessaires à l'identification du modèle d'une boîte noire. Ce même modèle servira à estimer la prochaine charge à injecter et à vérifier la condition d'arrêt de l'expérimentation automatique.

Nous présentons dans la suite la phase de l'identification du modèle de file d'attente d'une seule boîte noire. Cette identification se résume à l'identification des différents paramètres définissant une file d'attente T/X/K, en d'autres termes : la loi d'inter-arrivées T, la loi de service X et le nombre de serveurs K en fonction de la charge injectée à chaque palier. La discipline de service sera également inférée. Les tests en charge se feront, suivant le besoin, sur la boîte en isolation ou sur l'ensemble du système sous test. On se place alors à la fin d'une phase d'injection de charge et on suppose avoir les mesures nécessaires pour cette étape.

6.2 Description de l'identification du modèle

Revenons sur les travaux de Begin [**?**]. Ces travaux proposent d'élire un modèle parmi un ensemble de modèles en calculant une distance entre les points de la courbe des temps de réponse des modèles calibrés et les points de mesures. Cette approche peut répondre en partie à notre problème d'identification. Toutefois, elle présente une limite jugée déterminante pour l'identification du modèle : l'approche suppose l'existence d'un ensemble de modèles à partir desquels on choisit le meilleur modèle, c'est-à-dire le modèle qui réalise la distance minimale

par rapport aux mesures. Le problème se pose alors lorsque cet ensemble de départ ne contient que des modèles non adaptés, l'approche finira par choisir le modèle le moins mauvais.

Les travaux réalisés par Resnick et Reginald [81, 77] ont montré que le processus des inter-arrivées des requêtes dans les réseaux informatiques n'est pas toujours poissonien comme la plupart des travaux l'ont supposé. La recherche des distributions de service sur des requêtes d'un serveur 3-tiers ont montré que les distributions de service peuvent être des lois de type log-normales ou des lois Hyper-exponentielles. Ces types de distribution sont rares à tester dans des démarches d'identification de modèle. Alors qu'une telle erreur peut influencer énormément une solution de dimensionnement.

C'est pourquoi nous avons défini comme objectif d'explorer précisément l'ensemble des distributions d'inter-arrivées et de service qui peuvent modéliser notre système. Nous avons décomposé le problème de la modélisation à l'identification des distributions d'inter-arrivées et des distributions de service acceptables avant de déterminer et de calibrer le modèle de file d'attente adéquat à la boîte noire.

Pour déterminer les paramètres définissant un modèle de file d'attente, nous nous basons sur une étude statistique des mesures issues de l'injection de charge.

6.3 Identification d'une loi par analyse statistique

Avant de présenter l'identification de la loi des inter-arrivées et des services à partir d'un échantillon de mesures, nous exposons les différentes techniques d'identification et d'estimation d'un modèle en utilisant l'estimation et les tests statistiques.

Les mesures capturées et utilisées par notre processus d'identification sont des valeurs de temps de réponse et des dates d'arrivées. Par conséquent, on suppose que :

- les mesures des temps de réponse $r_1, \ldots, r_n$ sont n réalisations indépendantes d'une même variable aléatoire R de densité f_R.
- les mesures dates d'arrivées $a_1, \ldots, a_n$ sont n réalisations indépendantes d'une même variable aléatoire A de densité f_A.

L'objectif est de trouver la distribution d'inter-arrivées et de service. Il faut donc, à partir des mesures capturées, inférer un échantillon d'inter-arrivées et un échantillon de temps de service. Notons l'échantillon d'inter-arrivées à calculer $t_1, \ldots, t_n$, qui sont n réalisations indépendantes d'une même variable aléatoire T de densité f_T ; et l'échantillon des temps de service $x_1, \ldots, x_n$ qui sont n réalisations indépendantes d'une même variable aléatoire X de densité f_X. Ces valeurs de mesures doivent toujours être positives. Cela signifie que les lois à lesquelles on s'intéresse sont des lois à valeurs positives.

Il s'agit donc de déterminer, à partir d'un échantillon de mesures (inter-arrivées ou services), la forme de la (les) distribution(s) qui peu(ven)t convenir, ainsi que les paramètres de chaque distribution.

6.3.1 Identification par les statistiques descriptives

a Analyse graphique

La première étape par laquelle un expert commence une identification d'une loi théorique à un échantillon expérimentale est l'identification par les statistiques descriptives, c'est à dire en se basant sur l'utilisation d'histogrammes, indicateurs statistiques, etc. En effet, la comparaison de l'histogramme de l'échantillon à la courbe de densité d'une loi de probabilité de référence permet d'avoir une idée sur les familles des lois à vérifier. Dans la figure 6.1, il est clair que l'histogramme des mesures n'est pas très éloigné de la densité d'une loi exponentielle

$$f(x) = \lambda exp(-\lambda x)$$

. Par contre, on peut directement rejeter l'hypothèse qui rapproche cet histogramme à une densité d'une loi normale.

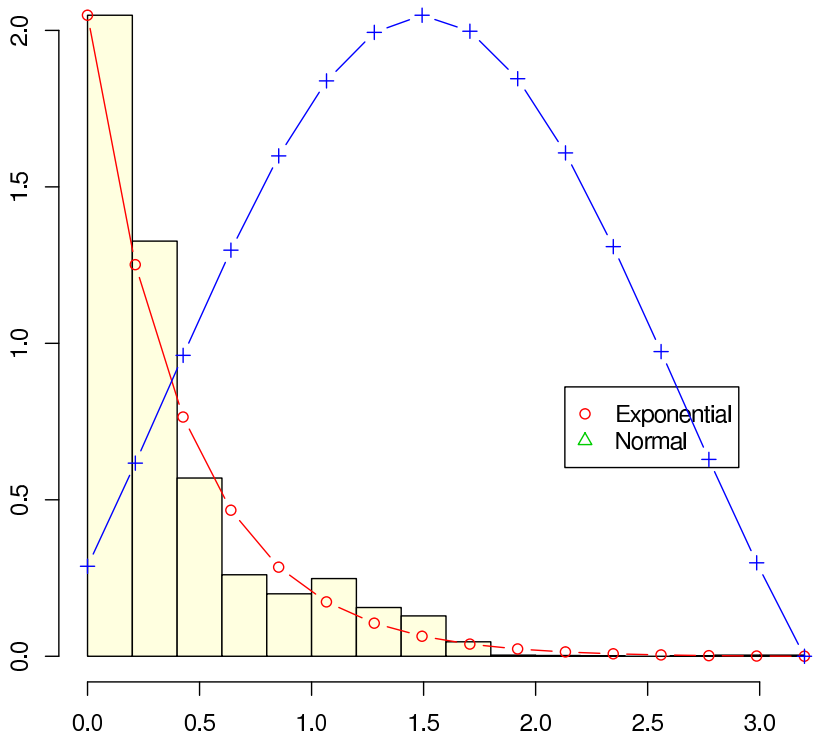

FIGURE 6.1 – Rapprochement de l'histogramme par la densité d'une loi de probabilité

Remarques 6.3.1. *L'identification visuelle peut se faire aussi avec la fonction de répartition empirique de la loi à identifier et de l'histogramme cumulé [70] : des outils tels que le graphe de probabilité [70, 94] permettent de facilité cette tâche de comparaison visuelle.*

La figure 6.2 montre la limite de l'identification d'une loi de probabilité par l'analyse graphique. En effet, dans ce cas, il est difficile de choisir entre la loi exponentielle et la loi lognormale.

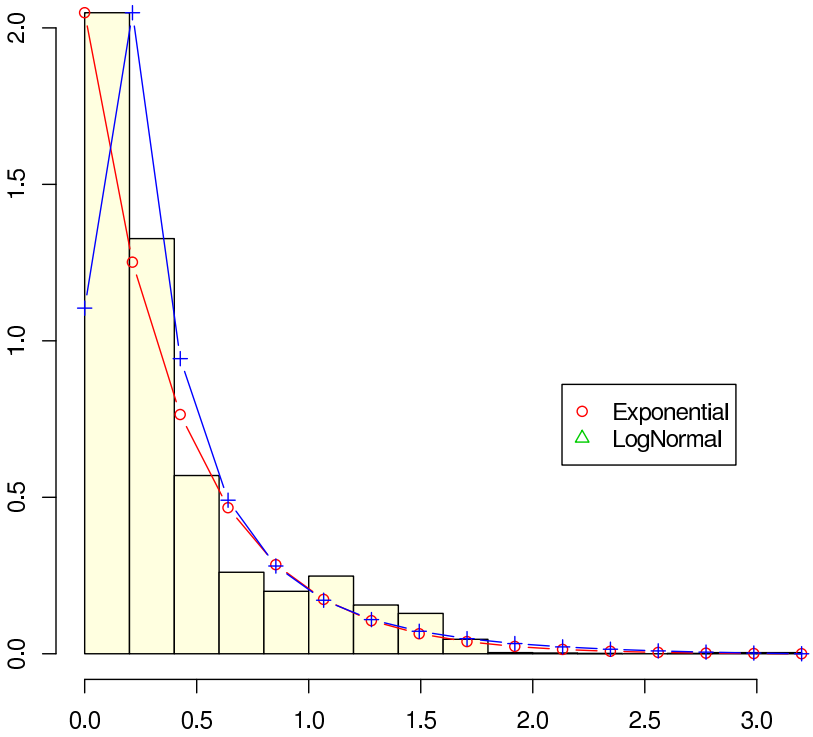

FIGURE 6.2 – Limites de l'identification graphique visuelle

b Analyse par indicateur statistique

La représentation graphique ne permet qu'une estimation visuelle non précise. Il est donc utile d'enrichir notre analyse par d'autres indicateurs statistiques quantitatifs qui vont nous aider à estimer la loi inconnue. On utilise généralement deux types d'indicateurs : des indicateurs de localisation (moyenne, médiane) et des indicateurs de dispersion (écart type, quantiles empiriques). Comment utiliser ces indicateurs ?

Prenons par exemple la loi exponentielle. Dans le cas de cette loi, l'espérance et l'écart type sont égaux à $1/\lambda$ et d'après la loi des grands nombres, la moyenne empirique converge vers l'espérance de la loi de probabilité. Cela nous permettra d'utiliser cette moyenne comme estimateur statistique pour vérifier l'adéquation de la loi exponentielle à notre échantillon. Toutefois, ces indicateurs restent des moyens d'identification non fiable, et cela à cause de leurs dépendances aux valeurs extrêmes. L'existence des valeurs aberrantes par exemple dans un échantillon décale la moyenne vers ces valeurs, ce qui fausse l'estimation de l'espérance empirique de la loi à tester.

L'utilisation de la statistique descriptive toute seule ne peut pas présenter un outil fiable pour l'identification d'un modèle de loi statistique pour un échantillon de mesures. Néanmoins, l'histogramme et les indicateurs statistiques permettent d'orienter l'expert vers une famille de lois à tester.

Remarques 6.3.2. *Dans notre procédure d'automatisation de l'identification des lois de service*

et des inter-arrivées, on utilisera les indicateurs statistiques qui orienteront nos prochaines recherches.

Comme précédemment expliqué, les paramètres de performances moyens d'une file M/G/1 ne dépendent que des deux premiers moments de la loi de service (moyenne et coefficient de variation). Par conséquent, il suffit d'ajuster la moyenne et le coefficient de variation d'une loi de service X quelconque pour obtenir les mêmes paramètres de performances moyens que la M/G/1. La moyenne ne pose pas de problème car elle peut prendre toute valeur positive (temps d'inter-arrivées et temps de service). Par contre, le coefficient de variation varie suivant la loi : il vaut 1 pour la loi exponentielle, 1/k pour une loi de type Erlang-k et il appartient à l'intervalle]0,1] pour une loi de type hypo-exponentielle-k. En se basant sur la valeur estimée du coefficient de variation et de son intervalle de confiance, on propose de sélectionner l'ensemble des lois à tester pour les inter-arrivées et le service. Cela réduira le coût des tests et vérification à faire par la suite. Ces résultats peuvent être utilisés aussi pour des files de type G/G/1 ou G/G/K puisque les approximations des paramètres de performance ne dépendent également que de la moyenne, du coefficient de variation et du nombre de serveurs K.

Pour vérifier précisèment l'adéquation à une loi, les tests d'hypothèse ont été utilisés.

6.3.2 Test d'adéquation ou d'hypothèse

Un test d'hypothèse est un test, dit d'ajustement, permettant d'accepter ou de rejeter une hypothèse statistique au vu d'un échantillon. Comme tout problème de décision, un test d'hypothèse cherche à trancher entre une hypothèse nulle H_0 et une hypothèse alternative H_1. Une seule hypothèse est à la fois correcte. Deux types d'erreurs sont alors possibles :

- Erreur de première espèce : rejeter à tort l'hypothèse H_0. La probabilité de cette erreur définit le niveau de signification du test.
- Erreur de seconde espèce : rejeter à tort l'hypothèse alternative H_1.

Ces deux erreurs varient d'une façon opposée. On ne pourra pas ainsi réduire ces deux erreurs à la fois. En pratique, on cherchera à réduire l'erreur la plus importante au sens du problème posé sans se soucier de l'autre.

La réponse du test d'hypothèse sera binaire : rejet ou non de l'hypothèse nulle H_0. On préfère souvent déterminer un seuil à partir duquel l'hypothèse H_0 sera rejetée : ceci définit la notion de **P-valeur** (**P-value**).

Définition 6.3.1. *Une **P-valeur** ou niveau de signification d'un test d'hypothèse est la probabilité seuil à partir de laquelle on rejettera l'hypothèse nulle, compte tenu de notre échantillon [83].*

Le calcul de cette probabilité est une pratique statistique courante permettant de déterminer si une hypothèse peut être acceptée avec une marge d'erreur fixée.

L'idée d'utiliser les tests d'hypothèse statistique pour l'identification d'une loi de probabilité revient à poser simplement comme hypothèse nulle « l'appartenance de l'échantillon à une même loi définie ». Ce test peut être fait sans donner d'estimation au paramètre de la loi. Cela revient à faire le test d'hypothèse pour une famille de lois. Il permettra ainsi un gain énorme sur le nombre de tests à effectuer et restreindra l'estimation des paramètres aux lois validées.

Plusieurs tests d'hypothèse existent [70, 90, 94, 57, 69, 29, 29, 95] (test de χ^2, test de Kolmogorov-Smirnov, test d'Anderson). Chaque test présente des spécificités, des avantages et des inconvénients que nous allons exposer en détail dans les paragraphes qui suivent.

6.3.3 Test de χ^2

Le test de χ^2 ou khi2 [90, 69, 57] est le test d'ajustement le plus connu. Son principe consiste à découper le domaine de la distribution en K classes disjointes. Dans chaque classe, on calcule à partir de la loi spécifiée sous l'hypothèse nulle H_0 la fréquence théorique attendue. On calcule ensuite le nombre de valeurs de notre échantillon N_i dans chaque classe (les fréquences d'observations). Puis, on détermine l'écart entre ces fréquences et les fréquences théoriques attendues. En comparant cet écart à un seuil, on décide d'accepter ou de rejeter l'hypothèse nulle H_0.

L'écart entre la loi de probabilité à tester et l'échantillon est mesuré à l'aide de la statistique :

$$Y_n = \sum_{i=1}^{k} \frac{(N_i - n.p_i)^2}{n.p_i}$$

avec p_i la probabilité pour qu'une observation appartient à la classe i.

Y_n tend vers une loi χ^2_{k-1} à (k-1) degrés de liberté, quand n tend vers l'infini. Pour appliquer le test de χ^2, il faut satisfaire la contrainte $n.p_i \geq 5$. Cela veut dire que chaque classe doit contenir au minimum 5 observations. Dans le cas contraire, il est conseillé de fusionner plusieurs classes adjacentes afin de respecter cette contrainte.

Avantages :

- Le test du χ^2 s'applique à toutes les distributions,
- Il est simple à mettre en œuvre.
- Lorsque les données sont constituées d'échantillons à valeur discrète, il est le seul test à pouvoir s'appliquer.
- Il est adapté aux échantillons de grande taille.

Inconvénients :

- Le test du χ^2 gaspille de l'information en regroupant les données en classes.
- Pour utiliser le test de χ^2, on est obligé de découper en classes d'une manière plus au

moins arbitraire. Le résultat du test peut dépendre de notre décomposition.

6.3.4 Test de Kolmogorov-Smirnov (KS)

Le principe de test de Kolmogorov-Smirnov [57, 69, 29, 90] consiste à mesurer l'écart maximum qui existe entre la fonction de répartition de la distribution théorique à tester et la densité cumulée de l'échantillon. Si cet écart dépasse un certain seuil, on rejettera l'hypothèse d'adéquation à l'échantillon de la loi théorique. Le test de Kolmogorov-Smirnov doit être utilisé quand la variable considérée a une distribution continue.

La distance de Kolmogorov-Smirnov s'écrit alors :

$$D = \max_x |F^*(x) - F(x)|$$

où $F(x)$ représente ... et $F^*(x)$ ets

La figure 6.3 montre la distance KS entre la loi normale et un échantillon.

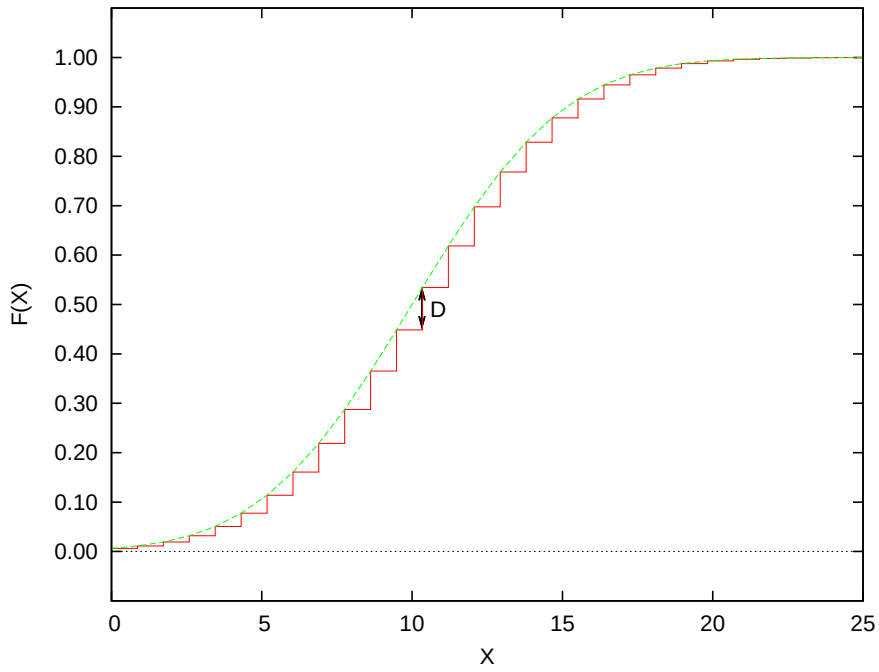

FIGURE 6.3 – Distance de Kolmogorov-Smirnov

Avantages :
- Le test de Kolmogorov-Smirnov s'applique à toutes les distributions continues.
- L'avantage du test de KS par rapport à un test du χ^2 est qu'on n'a pas recours à une répartition arbitraire en classes (intervalles).

Inconvénients :
- Ce test représente une sensibilité particulière aux valeurs aberrantes. Il suffit d'avoir pour une seule valeur une grande différence entre l'échantillon et la distribution théorique pour que le test soit rejeté, puisque la distance sera égale à ce maximum.
- Il est plus adapté à l'échantillon de petite taille (entre 30 et une centaine de valeurs).
- Le test de Kolmogrov-Smirnov est moins adapté que le test d'Anderson-Darling aux familles de distributions dites « à queue lourde ».

6.3.5 Test d'Anderson-Darling (AD)

Ce test est considéré comme une amélioration du test de Kolmogorov-Smirnov [29]. En effet, la valeur critique de l'acceptation de la loi théorique varie en fonction de la loi à tester. Cela présente un avantage puisque chaque valeur critique prendra en compte des particularités de la loi et un inconvénient car on doit refaire les calculs des valeurs critiques pour chaque distribution. Les tableaux des valeurs critiques sont actuellement disponibles pour la loi normale, log-normale, exponentielle, Weibull, extrême type I et logistique. Un coefficient correctif peut être ajouté à la statistique d'Anderson-Darling prenant en compte la taille n de l'échantillon.

La statistique A^2 calculé pour le test d'Anderson peut s'écrire comme suit :

$$A_n^2 = -n - \frac{1}{n}\sum_{i=1}^{n}(2i-1)[lnF(X_i) + ln(1 - F(X_{n-i+1}))]$$

Avec F : la fonction de répartition de l'échantillon et n la taille de l'échantillon. Le test d'Anderson-Darling accorde plus d'importance à la queue de la distribution. Cela justifie sa grande utilisation avec les lois à queue lourde. Pour notre procédure d'identification de modèle et comme cité dans le chapitre 5, il est très important d'avoir des données fiables sur la queue de la courbe des temps de réponse qui représentent pour nous la région où le point de saturation recherché se trouve.

Avantages :
- Le test d'Anderson-Darling s'applique aux lois : normale, log-normale, exponentielle, Weibull...
- Il est adapté au distributions à queue lourde.
- Il améliore la distance de Kolmogorov-Smirnov ; le calcul de la distance dépend de la loi à tester.

Inconvénients :
- Ce test ne peut être utilisé qu'avec un nombre très réduit de distributions dont on dispose des distances.

6.3.6 Autres tests

Plusieurs autres tests d'ajustement ont cherché à répondre aux faiblesses des tests d'ajustement classiques. Parmi ces tests, le test de Cramer-von Mises [29, 90] qui cherche à réduire la vulnérabilité du test de Kolmogorov-Smirnov aux valeurs aberrantes.

Pour cela, ce test propose de calculer sa statistique en se basant non seulement sur le maximum des écarts comme le test de Kolmogorov-Smirnov, mais aussi sur la somme des écarts entre la fonction de répartition empirique et la fonction de répartition de l'échantillon. Cela réduit énormément la sensibilité du test aux valeurs aberrantes.

$$I = \int_{-\infty}^{+\infty} [F^*(x) - F(x)]^2 dF(x)$$

La statistique de Cramer-von Mises est définie alors par l'intégrale sur le carré de l'écart entre la fonction de réparation empirique et celle de l'échantillon.

6.4 Identification de la loi d'inter-arrivées

Comme cité au début de ce chapitre, notre approche consiste à décomposer l'identification du modèle d'une boîte noire en une identification de la loi d'inter-arrivées, de la loi de service et de l'identification du nombre de serveurs. Dans une application répartie, les inter-arrivées au niveau d'une boîte noire (composant du système sous test) dépendent des inter-arrivées des requêtes qui arrivent au système et des services des boîtes en amont de notre boîte. Dans un premier temps, nous cherchons à déterminer un échantillon des inter-arrivées. Puis, nous identifions la loi d'inter-arrivées qu'on validera à la fin.

6.4.1 Capture des requêtes en entrée et échantillonnage

Afin de capturer les inter-arrivées des requêtes en entrée d'une boîte, nous procédons à une injection de charge au niveau de l'entrée du système. Puis, nous utilisons les fichiers logs qui journalisent les informations inhérentes au déroulement des requêtes au niveau de chaque boîte. De ces fichiers, on récupère les dates d'arrivée des requêtes et on calcule les inter-arrivées correspondantes.

6.4.2 Analyse de l'échantillon

Pour déterminer automatiquement la forme de la distribution de l'échantillon des inter-arrivées, nous utilisons une approche basée sur les tests d'hypothèse statistiques. Ces tests

d'hypothèse permettent de sélectionner quelles distributions peuvent convenir à l'échantillon analysé. Le choix de test statistique dépend du type de la loi à tester et à l'échantillon à analyser (taille, type). Dans notre cas, on utilisera dans un premier temps le test de Kolmogorov-Smirnov, en raison de son adéquation aux distributions à temps continu. Puis, on complètera, par un test d'Anderson-Darling lorsqu'on traite des lois de type queue lourde d'une part, et d'autre part, lorsque la distance associée à cette loi existe. Toutefois, ces tests sont appropriés aux échantillons de petite taille. Comme nos échantillons de mesures sont de grande taille, nous choisissons de manière uniforme un sous-échantillon de petite taille de l'échantillon à analyser (100 mesures). Sur cet échantillon est effectué le test statistique. Les distributions résultantes ayant une p-valeur plus grande que 0.1 pour le test de Kolmogorov-Smirnov et plus grande que 0.5 pour le test d'Anderson-Darling [83] sont sélectionnées comme les distributions possibles de notre échantillon d'inter-arrivées.

Le test de Kolmogorov-Smirnov et Aderson-Darling peuvent vérifier l'adéquation d'un échantillon à une famille de distributions, connues dans la littérature comme des distributions apparaissant naturellement dans plusieurs systèmes [77] : La famille de distributions exponentielles, les distributions à queue lourde, etc. Nous commençons par vérifier une distribution de la famille exponentielle. Pour cela, nous calculons le coefficient de variation CV^2 de l'échantillon ainsi que son intervalle de confiance. Selon la valeur du CV^2, nous testons un sous-ensemble de distributions :

- Si l'intervalle de confiance du CV^2 contient 1, nous testons la distribution exponentielle.
- Si CV^2 appartient à]0, 1[, nous testons la distribution hypoexponentielle(k), la distribution d'Erlang(k) et la distribution gamma $\Gamma(a, \lambda)$ telque $a > 1$.
- Si CV^2 appartient à]1,+∞[, nous testons la distribution hyperexponentielle(k), la loi uniforme, la loi normale, la lognormale et la distribution de Weibull.

Pour chaque distribution d, nous analysons un échantillon comme suit :
- Si nécessaire, nous portons certaines transformations sur l'échantillon (tel qu'un décalage).
- Nous estimons les paramètres de la distribution d en utilisant un estimateur du Maximum de vraisemblance.
- Nous choisissons un sous-échantillon de petite taille. Cette taille peut être prise entre 30 et 100 valeurs.
- Nous appliquons le test statistique pour vérifier la distribution d avec les paramètres estimés. Le test est répété plusieurs fois (La loi des grands nombres en statistique différentielle pour la détermination d'une loi de probabilité à partir d'une série d'expériences). Nous avons opté pour 1000 expériences par test. Ensuite, une moyenne (nombre de test qui ont abouti) des p-valeurs obtenues sont retenues comme p-valeur d'acceptation de la distribution.
- Nous retenons uniquement les distributions dont le test statistique donne une p-valeur supérieure au seuil défini précédemment.

6.5 Identification de la loi de service

La loi de service est identifiée à partir de l'analyse des mesures de performance délivrées lors de l'injection de charge soumise à la boîte noire. Tel qu'expliqué dans le chapitre 5, plusieurs paliers d'injection sont effectués, délivrant plusieurs ensembles de mesures à analyser. Nous nous intéressons dans notre travail à deux types de mesures à chaque palier i :

- Les temps de réponse, notés R_k, $1 <= k <= N_i$, N_i étant la taille de l'échantillon du palier i.
- Les dates d'envoi des requêtes A_k, $1 <= k <= N_i$.
- L'utilisation des ressources U. Une ressource U représente toute ressource physique utilisée par la boîte noire, telle que le processeur (CPU), la mémoire, le disque, le réseau, etc.

A partir de ces mesures, un échantillon des temps de service est calculé, puis analysé pour déterminer les distributions statistiques qui lui conviennent.

6.5.1 Calcul de l'échantillon des temps de service

A partir des temps de réponses et des temps d'inter-arrivées (déduits des dates d'arrivées des requêtes), nous calculons un échantillon des temps de service correspondant à chaque palier d'injection.

Soient (X_k), $1 <= k <= N_i$ les temps de service à calculer, (R_k),$1 <= k <= N_i$ les temps de réponse capturés et (t_k), $1 <= k <= N$ les durées d'inter-arrivées. Dans ce contexte, nous nous basons sur des résultats de Kleinrock [50] reliant les temps de service, les temps de réponse et les temps ou taux d'inter-arrivées, et qui diffèrent selon la politique de service adoptée dans le système et le nombre de serveurs identifiés du modèle.

Cas d'une file d'attente G/G/1/FIFS [50] :

$$R_k = [R_{k-1} - t_k]^+ + X_k$$

Ce résultat a été énoncé pour un modèle utilisant un seul serveur et une politique de service FIFS. Dans le cas où une politique de service différente est utilisée, une équation similaire peut être déduite.

Cas d'une file d'attente G/G/k/FIFS :

Quant au cas de plusieurs serveurs, il est nécessaire de redéfinir l'équation précédente, qui ne reste plus valable puisqu'à un palier donné d'injection de charge, il est possible d'identifier un modèle de file d'attente caractérisé par un nombre K de serveurs. Pour généraliser ce résultat, nous proposons le résultat similaire suivant :

$$R_k = [R_j - t_{j,k}]^+ + X_k$$

Où j correspond à la requête précédente qui a libéré le serveur ayant servi la k^{ime} requête, et $t_{j,k}$ est l'inter-arrivée entre la j^{ime} et la k^{ime} requête. La j^{ime} requête est déterminé en recalculant itérativement les temps de service X_k, $1 <= k <= N$, en commençant à partir de la première requête servie et en utilisant les R_j et $t_{j,k}$ calculés à partir des mesures collectées.

Cas d'une file d'attente G/G/1/RR ou PS :

L'estimation du temps de réponse en fonction du temps de service est donnée dans [49] par la formule :

$R(x) = x/(1-\rho)$ où $\rho = \lambda.\bar{x}$

Pour estimer x on doit utiliser la valeur de $\bar{x}$ (utilisation du théoreme de point fixe [3])

Remarques 6.5.1. *Le calcul utilise une formule qui dépend de la discipline de service supposée. Cette discipline est une hypothèse qui sera rejetée en cas d'une forte incohérence dans le temps de service généré. En effet, une valeur de temps de service est incohérente lorsqu'elle est négative ou lorsqu'elle est supérieure au temps de réponse. Si le nombre de valeur incohérente dépasse un certain seuil, l'hypothèse est rejetée et on utilise une autre discipline. Cette hypothèse peut être également rejeté à la validation du modèle.*

6.5.2 Analyse statistique de l'échantillon des temps de service

L'analyse de l'échantillon des temps de service se fait de la même manière que celle appliquée à l'échantillon des inter-arrivées (voir section 6.4). Nous commençons donc par l'épuration de notre échantillon en éliminant les valeurs aberrantes (5 ou 10% des valeurs qui sont excessivement éloignées du reste des valeurs)[94].

D'éventuelles transformations peuvent être portées à l'échantillon avant le test d'hypothèse (par ex., décalage). Ceci peut être expliqué autrement par rapport à l'échantillon des temps des inter-arrivées : Alors que pour les temps d'inter-arrivées, cela peut être expliqué par un temps d'attente constant avant l'arrivée au système (comme par exemple lorsqu'un mécanisme de gestion d'accès est utilisé), ce décalage, dans le cas des temps de service, peut être interprété par un temps de traitement constant. Dans la suite, nous montrons un exemple qui illustre notre approche d'identification des lois des temps d'inter-arrivées et des temps de service.

6.6 Exemple

Dans cette section, nous allons présenter l'identification du modèle d'une boîte noire en appliquant les tests statistiques pour la loi de service et la loi des inter-arrivées. Les figures 6.4 et 6.5 montre la représentation graphique des 2 échantillons : services, inter-arrivées. Cette expérimentation est réalisée avec l'application Rubis presentée dans la section 4.5. l'application est déployé sur

6.6.1 Identification de la loi des inter-arrivées

Nous allons analyser un échantillon d'inter-arrivée d'une expérimentation d'injection de charge par paliers (voir chapitre 5). L'histogramme et le graphe de probabilité de cet échantillon sont décrits dans la figure 6.4. L'échantillon des temps d'inter-arrivées a un coefficient de variance CV^2 égal à 1.01, avec un intervalle de confiance de $[0.95, 1.06]$ (taux d'erreur de 5%). Cette valeur de CV^2 indique que la loi exponentielle est une loi possible pour notre identification. Nous avons d'abord estimé le paramètre de la distribution exponentielle λ à 30.34. Puis, nous avons utilisé le test de Kolmogorov-Smirnov sur un échantillon de taille 100, choisi au hasard à partir de l'échantillon initial. La p-valeur obtenue est de 0.59. Par conséquent, nous ne rejetons pas l'hypothèse d'une distribution exponentielle.

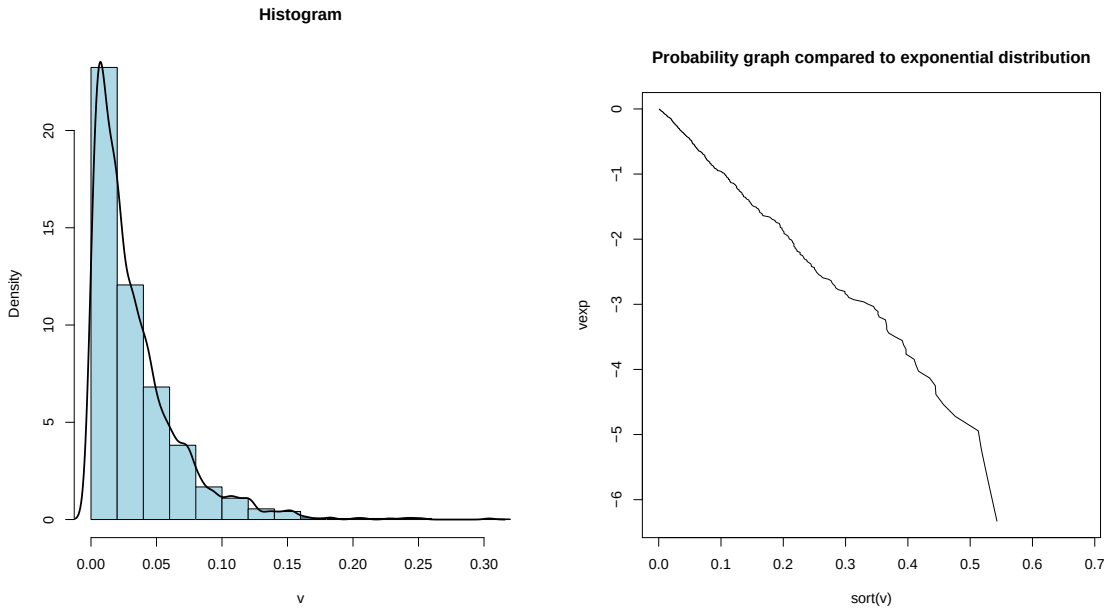

FIGURE 6.4 – Analyse Graphique de l'échantillon des inter-arrivées : histogramme et graphe de probabilité

Remarques 6.6.1. *Cette expérimentation représente une première validation de notre approche. En effet, le modèle exponentiel que nous avons identifié n'est que le résultat de l'injection de*

charge que nous avons injecté nous même à notre boîte noire. Cela représente aussi une validation à notre système d'injection de charge.

6.6.2 Identification de la loi de service

Dans cette expérimentation, nous avons injecté la boîte noire avec des requêtes HTTP et des temps d'inter-arrivées de loi exponentielle. Après collecte et épuration de nos échantillons, nous avons appliqué notre méthodologie d'identification de la loi de service. L'application a atteind la saturation apres 13 paliers d'injections et une charge maximale de 125 virtual users/s. Le taux de service initiale μ_0 a été estimé à 46,97 requêtes/s.

La figure 6.5 présente les histogrammes des temps de service calculés correspondant à différents niveaux d'injection de charge. L'allure des courbes nous laisse penser que la loi exponentielle ne correspondra à aucun échantillon, alors que le tableau des résultats des tests statistiques ne rejette pas la loi exponentielle. Cela est bien confirmé par les valeurs CV^2 estimées. D'autres lois telles que la loi log-normale, la loi Weibull, et la loi hyper-exponentielle ont été également validées par nos tests statistiques. Ceci peut être expliqué par le fait que la valeur du CV^2 estimée est très proche de 1, ce qui représente un point d'intersection des intervalles de confiance du CV^2 des lois empiriques à tester.

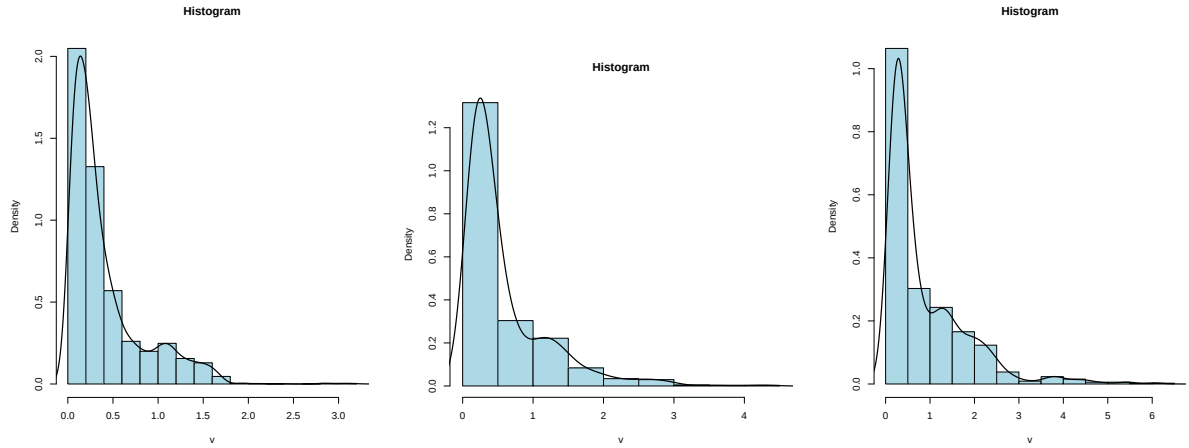

FIGURE 6.5 – Analyse graphique de l'échantillon de service : histogrammes des étapes 8,10 et 13

6.6.3 Identification du nombre de serveur

Comme expliqué dans le chapitre 5, le nombre de serveurs du modèle de file d'attente est obtenu expérimentalement à partir de la valeur de la charge maximale de saturation C_{max}, déterminée expérimentalement et la valeur initiale C_{max0} déterminée à partir de l'injection d'une

Num exp	Charge	CV^2	Moyenne	95% IC	Type de distributions sélectionnées
2	20	0.04	0.02	[0.21, 0.23]	Log-Normal,Gamma
6	60	2.18	0.08	[1.39, 1.55]	Log-normal
7	70	1.96	0.19	[1.33, 1.46]	Expo,Hyper-expo,Log-Normal,Gamma,Weibull
8	80	1.03	0.40	[0.97, 1.05]	Expo,Hyper-expo,Log-Normal,Gamma,Weibull
10	100	1.07	0.58	[0.99,1.07]	Expo,Hyper-expo,Log-Normal,Gamma,Weibull
13	125	1.11	0.87	[1.00, 1.10]	Expo,Hyper-expo,Log-Normal,Gamma,Weibull

TABLE 6.1 – Analyse d'un échantillon de temps de service

Num exp	Exponentielle	Hyper-exponentielle	Log-Normal	Gamma	Weibull
2	0.00	0.00	**0.16**	**0.14**	0.08
6	0.07	0.03	0.21	0.07	0.05
7	**0.11**	**0.23**	**0.29**	**0.17**	**0.20**
8	**0.44**	**0.43**	**0.47**	**0.41**	**0.41**
10	**0.32**	**0.29**	**0.39**	**0.29**	**0.31**
13	**0.17**	**0.29**	**0.34**	**0.21**	**0.23**

TABLE 6.2 – Résultats des P-valeurs pour chaque palier d'injection

requête à vide. Après 27 minutes du début de l'expérience, nous avons atteint le 12ieme palier d'injection avec 125 Vusers. La processeur atteint une utilisation de 96%. Cela montre que le processeur est la ressource responsable de la saturation. A la fin de expérimentation, l'identification de nombre de serveur indique un nombre de serveur k=3.

6.7 Modèle global de la boîte noire

La combinaison des lois des inter-arrivées, de service et du nombre estimé de serveurs constitue un modèle global possible pour notre boîte noire. Néanmoins, ce modèle à besoin d'être validé. Cette validation se fera en comparant les résultats d'indices de performance calculés théoriquement pour chaque modèle avec les résultats empiriques, quelques soit la nature de ces résultats : exactes (temps de réponse, utilisation, du modèle M/G/1) ou approchés (borne supérieure du temps d'attente dans le cas de la G/G/1 ou la G/G/K).

Lorsque plusieurs modèles sont validés, notre système proposera tous les modèles validés à l'utilisateur. Dans ce cas, l'utilisateur aura la possibilité d'utiliser un de ces modèles. Cela représente bien évidement une richesse pour notre approche, puisqu'elle laisse plus de choix pour les prochaines phases de dimensionnement. L'utilisateur peut dans ce cas utiliser le modèle qui simplifie le plus le calcul du réseau de files d'attente, ou le modèle le plus proche par rapport à ces mesures pour expliquer des phénomènes locaux à la boîte (par exemple une saturation au niveau d'une boîte).

6.8 Checklists d'identification d'une loi

Identification d'une loi dans un contexte donné

1. **Traitement préléminaire de l'échantillon** : élimination des valeurs aberrantes.
2. **Contrôle de la taille de l'échantillon** : vérifier si le nombre de mesures collectées $N >= \frac{100.z}{\epsilon * Mean(R)}^2 * Var(R)$, avec z=1.96 (intervalle de confiance 95%) et ϵ=0.001, **sinon sortie** : Statistique non significative.
3. **Présélection des familles de lois à tester** : calculer le CV^2 et son intervalle de confiance
4. Suivant l'intervalle de confiance du CV^2, sélectionner les distributions L à tester.
5. **Choix du test d'hypothèse** : Pour chaque loi L, choisir le test d'hypothèse adéquat à réaliser : χ^2, KS ou AD.
6. **Estimation des paramètres des distributions à tester** à partir de l'échantillon initial de taille N en utilisant le maximum de vraisemblance.
7. **Sous-échantillonage** : sélectionner un sous-échantillonage de taille $\tilde{N}$ (30 à 100 valeurs).
8. **Réalisation du test d'hypthèse** : appliquer le test de KS ou AD pour le sous-échantillon obtenu et les paramètres estimés des distributions.
9. Refaire le sous échantillonage et le test d'hypothèse plusieurs fois (de 100 à 1000 fois).
10. Sélectionner les lois dont la P-value est supérieure au seuil de sélection (0.1 pour le KS).

Identification des lois de service et des lois d'inter-arrivées en fonction du contexte

1. Injecter 1 seule requête à la boîte noire.
2. Estimer $Cmax_0 = \mu_0 = \frac{1}{X_0}$ en utilisant le modèle de file d'attente M/G/1.
3. Déterminer la prochaine charge à injecter C_i suivant la politique d'injection utilisée.
4. Collecter les mesures des temps de réponses, d'arrivées et d'utilisation.
5. Extraire les temps de service et les temps d'inter-arrivées.
6. Estimer le temps de stabilisation du palier courant.
7. Éliminer les valeurs des temps de service et des temps d'inter-arrivées appartenant à la période d'instabilité.
8. Appliquer les tests d'hypothèse statistiques (cheklist 1).
9. Déterminer les distributions des lois de service et des inter-arrivées et le nombre de serveurs.
10. Déterminer la nouvelle valeur de Cmax si le nombre de serveurs change.
11. Valider les modèles obtenus en comparant les résultats théoriques aux mesures.
12. Vérifier la condition $C_i < Cmax$ et l'utilisation U_l de chaque ressource l, sinon refaire à partir de l'étape 3.

6.9 Conclusion

Ce chapitre a présenté dans un premier volet les différents tests statistiques permettant d'identifier la loi empirique à partir d'un échantillon et en présentant l'avantage et l'inconvenient de chaque test. Cette étape nous a permis de choisir les tests le plus adaptés à notre approche d'identification de la loi des inter-arrivées et la loi de service. En combinant ces lois avec le nombre de serveurs estimés dans le chapitre précédent, nous avons obtenus les modèles de file d'attente qui peuvent représenter notre boîte noire sous test à différents paliers de charge. Seuls les modèles qui seront valides par comparaison entre leurs résultats théoriques et les mesures seront sélectionnées. Ces derniers représenteront alors un point d'entrée pour les algorithmes de résolution des réseau de files d'attente, utilisés afin de dimensionner un système global tel qu'il est décrit dans le chapitre 4. Tous les modèles possibles peuvent participer également à l'identification et à l'explication des problèmes (saturation . . . ,etc.) locaux au niveau d'une seule boîte noire.

Le chapitre suivant détaillera la façon avec laquelle nos modèles de boîtes noires seront utilisés afin d'aider à dimensionner un système composé d'un réseau de boîtes noires.

Chapitre 7

Construction du modèle global et dimensionnement

Introduction

Les deux chapitres précédents ont présenté une approche complète allant de l'expérimentation à l'identification du modèle de file d'attente associé à une boîte noire. Conformément à l'approche globale décrite par la figure 4.1, ce même processus est appliqué sur l'ensemble des boîtes formant le système sous test. L'ensemble de ces modèles constitue alors le point d'entrée à la phase de dimensionnement qui sera traité dans ce chapitre. Celui-ci présente d'abord comment combiner différents modèles de boîte noire afin de produire un modèle global et d'étudier ses performances. La deuxième partie de ce chapitre est dédiée au dimensionnement par analyse ou par simulation suivant le modèle global obtenu et la qualité du dimensionnement souhaité. Une dernière partie met en évidence la résolution du problème de déplacement de goulets d'étranglement.

7.1 Choix du modèle de la boîte à utiliser

Après l'identification du modèle, chaque boîte noire dispose d'un ou plusieurs modèles différents fonctions de la charge injectée. En effet, les expérimentations réalisées sur plusieurs systèmes ont montré que la charge en entrée joue un rôle capital dans le choix du modèle. Ainsi, toute étude de performance rigoureuse, doit définir l'intervalle de niveau de charge avec lequel cette étude doit être réalisée et, en conséquence, on fixe le modèle de file d'attente et les paramètres à utiliser.

Le choix du modèle, lorsque plusieurs modèles de la boîte sous test sont possibles, dépend de la méthode de résolution utilisée pour le réseau de file d'attente. En fonction du type de l'ap-

proche, par simulation ou par une des approches analytiques, exacte ou approchée, on choisit le modèle de boîte noire approprié. Une résolution par l'algorithme exacte de type MVA ou par réseau de Jackson contraint le choix en cherchant un modèle markovien pour chaque boîte formant le réseau de file d'attente. Réciproquement, la disponibilité du type du modèle souhaité pour la charge d'étude définie impose aussi le type de l'approche à utiliser.

Finalement, la forme du réseau participe aussi dans le choix du modèle de la boîte à utiliser. Le choix de la configuration et la nature des interactions définissent la forme du réseau de file d'attente (ouvert, fermé ou mixte) qui limite les approches de résolution possibles et par conséquent le modèle à choisir.

Dans l'exemple 6.6, les charges supérieures à 70 requêtes/s proposent plusieurs modèles (M/M/3, M/HyperExpo/3, M/LogN/3, M/Gamma/3, M/Weib/3) avec des qualités équivalentes. L'étude des performances pour une charge supérieure à 70 requêtes par seconde donne plusieurs possibilités pour les méthodes de résolution. L'algorithme MVA est sélectionné pour un modèle M/M/k. Si l'on procède par simulation, on choisit le modèle le plus précis par rapport à son p-value (voir tableau 6.2) ou en fonction de la durée de simulation souhaitée.

7.2 Composition des modèles de boîtes noires

La composition des modèles pour construire le modèle global est fonction de l'objectif de l'étude. En effet, l'évaluation des performances se fait soit pour étudier la configuration (architecture du réseau des boîtes noires) en cours du système sous test pour qualifier le système réel, soit pour évaluer une reconfiguration éventuelle du système pour optimiser ou réparer un problème (dégradation des performances, goulets d'étranglement).

- **Cas de la qualification d'un système réel** : Dans ce cas, le modèle global est entièrement défini par les files d'attente associées à chaque boîte et par la matrice de routage entre différentes files. Soit $P = ((p_{i,j}))$ cette matrice de routage avec $p_{i,j}$ la probabilité pour qu'une requête sortante de la boîte i soit adressée à la boîte j. Pour calculer ces probabilités, on analyse les traces d'exécution du trafic entrant et sortant au niveau de chaque boîte. Durant une exécution faisant intervenir tout le système sous test, on calcule le nombre de requêtes entrantes et sortantes au niveau de chaque boîte. Des ratios de trafic sont alors calculés pour définir les probabilités de routages entre les files d'attente.
  Prenons l'exemple d'un système S formé par trois boîtes noires A, B et C (voir figure 7.1), où A est lié à B et C. Dans ce cas, nous calculons le nombre de requêtes sortantes de A et le nombre de requêtes reçues par chaque station de B et C. Supposons qu'il y ait 18000 requêtes sortantes de A, 10000 requêtes entrantes pour B et 8000 pour C. Les probabilités de routage sont alors $p_{a,b} = 10000/18000 = 0,56$ et $p_{a,c} = 8000/18000 = 0,44$, d'où la matrice de routage décrivant toutes les interactions du réseau de file d'attente.
- **Cas de l'évaluation d'une reconfiguration alternative** : Dans ce cas, l'expérimentateur fournit tous les paramètres composant le réseau de files d'attente à étudier. Il définit la matrice de routage qui décrit les interactions entre les boîtes noires et les paramètres des

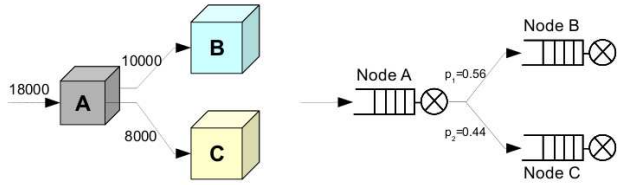

FIGURE 7.1 – Exemple d'interconnexion de boîtes noires

modèles des boîtes utilisées. Après l'analyse du réseau et la détermination des paramètres de performances globales, la reconfiguration proposée est soit validée, soit rejetée. En cas de rejet (ne satisfait pas le SLO défini), une autre reconfiguration est évaluée jusqu'à l'obtention d'une configuration valide répondant à la qualité de service souhaitée.

7.3 Validation du modèle global

La validation se fait en comparant les mesures des performances empiriques collectées dans la première phase de l'expérimentation à leurs estimations théoriques calculées ou simulées à partir du modèle. Cette validation se fait en deux phases :

- La première phase est la validation de tous les modèles des boîtes noires. Utilisés dans composition du modèle global (voir section 6.7).
- La deuxième phase est la validation du réseau de file d'attente représentant le système global. Cette validation n'a d'intérêt que pour la qualification d'un système existant. Dans le cas d'évaluation d'une configuration alternative, ne disposant pas le système réel, on ne pourra valider que des portions du modèle qui correspondent à des parties existantes. Le procédé de comparaison est identique à la première phase mais en utilisant des mesures globales : des mesures de temps de réponses moyens et/ou de nombres de clients sont comparées à ces mêmes paramètres calculés à partir du modèle de réseaux de file d'attente.

Le temps de calcul des paramètres de performances théoriques nécessaire à la validation dépend de la complexité du réseau de file d'attente généré et des méthodes de résolution ou de simulation utilisées. Cette deuxième phase de validation peut se faire en parallèle avec l'étape d'analyse (étape 4, figure 4.1). Cette optimisation permet un gain de temps conséquent surtout dans le cas d'une analyse par simulation.

7.4 Exemple de modélisation et de validation

Pour illustrer les différentes étapes de notre processus de modélisation, nous avons réalisé une expérimentation complète allant de l'identification des boîtes jusqu'à la validation du

modèle global pour l'application trois tiers *SampleCluster*. L'application est déployée sur un serveur Tomcat comme conteneur Web, un conteneur EJB (Enterprise Java Beans 3.0) et une base de données MySQL (voir la figure 7.2).

7.4.1 Décomposition

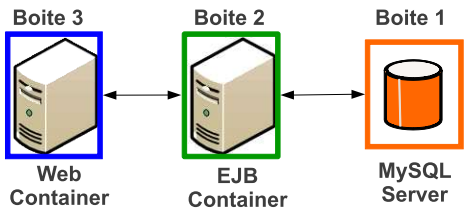

FIGURE 7.2 – Décomposition de l'application *SampleCluster*

Le système est décomposé en trois boîtes noires suivant son architecture multi-tiers : une première boîte encapsule le conteneur Web, une deuxième le conteneur EJB et la troisième boîte la base de données (voir figure 7.2). Cette décomposition est pertinente pour un dimensionnement d'une application multi-tiers puisque, dans la pratique, on peut déployer les tiers sur des ensembles de ressources séparées. Ainsi on peut détecter le ou les tiers responsables du goulet d'étranglement, et optimiser les tiers défaillants (réplication, re-configuration . . .).

7.4.2 Paramètrage

Chacune des boîtes noires est modélisée en utilisant notre processus d'identification automatique. On utilise CLIF pour injecter la charge, avec un scénario d'injection de charge et avec des utilisateurs virtuels ayant des comportements considérés comme réalistes (obtenus à partir de capture de session réelle). La latence du réseau est considérée comme négligeable pour ces expériences, en comparaison avec les temps de réponse qui sont de l'ordre de quelques millisecondes. En effet, l'ensemble des machines est connecté via un équipement réseau Ethernet dont la latence est de l'ordre de quelques μs. Pour obtenir des mesures d'utilisation des ressources et détecter une saturation du système, on utilise les sondes CLIF (processeur, mémoire vive et JVM). On définit les seuils de bon fonctionnement des tiers à 80% du processeur, 80% de la mémoire vive et 5% pour la mémoire JVM libre.

7.4.3 Isolation des boîtes noires

Pour modéliser une boîte noire, on doit d'abord procèder à son isolation. L'isolation du tiers base de données est la plus simple, puisqu'il est le dernier tiers et qu'il reçoit les requêtes uniquement du tiers EJB (voir figure 7.3) . L'isolation des tiers Web et EJB exige plus de travail. En effet, l'isolation peut se faire soit en développant des plugins RMI et SQL (voir section

4.6), soit en utilisant la méthode expérimentale pas à pas permettant de remplacer chaque tiers modélisé par son modèle afin de déduire les temps de réponse associés au tiers suivant. Les figures 7.4 et 7.5 montrent l'utilisation de cette dernière méthode pour la extraction du temps de réponse au niveau du conteneur EJB et du conteneur Web.

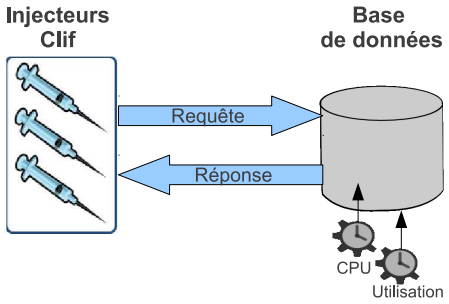

FIGURE 7.3 – Isolation de la boîte 1 : base de données

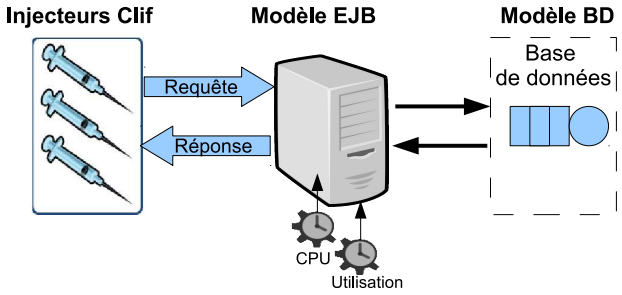

FIGURE 7.4 – Isolation de la boîte 2 : conteneur EJB

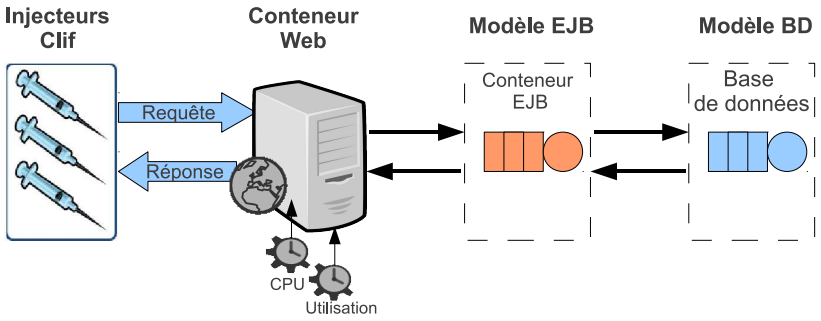

FIGURE 7.5 – Isolation de la boîte 3 : conteneur Web

7.4.4 Modèle de la base de données

L'expérimentation sur la boîte noire « base de données » est realisée sur un serveur Linux avec deux processeurs PIII 1,4 GHz, et 1 Go de RAM. Nous avons déployé les injecteurs de charge Clif sur un serveur Linux bi-processeur Xeon 2,8 GHz et 2 Go de RAM.

Charge	Modèles identifiés	Modèles selectionné	Paramètres
1	$M/\Gamma/4$, M/LN/4, M/Norm/4, M/Wbl/4	M/LN/4	m=-8.010, sigma=0.089
6	$M/\Gamma/4$, M/LN/4, M/Norm/4, M/Wbl/4	M/LN/4	m=-8.034, sigma=0.276
11	$M/\Gamma/4$, M/LN/4, M/Norm/4, M/Wbl/4	M/LN/4	m=-8.051, sigma=0.333
16	$M/\Gamma/4$, M/LN/4, M/Norm/4, M/Wbl/4	M/LN/4	m=-8.020, sigma=0.396
21	$M/\Gamma/4$, M/LN/4, M/Norm/4, M/Wbl/4	M/LN/4	m=-8.030, sigma=0.421
26	$M/Hr_2/4$, $M/\Gamma/4$, M/LN/4, M/Norm/4, M/Wbl/4	M/LN/4	m=-8.053, sigma=0.428
31	$M/Hr_2/4$, $M/\Gamma/4$, M/LN/4, M/Norm/4, M/Wbl/4	M/LN/4	m=-8.033, sigma=0.464
36	$M/Hr_2/4$, $M/\Gamma/4$, M/LN/4, M/Norm/4, M/Wbl/4	M/LN/4	m=-8.074, sigma=0.476
45	M/M/4, $M/Hr_2/4$	$M/Hr_2/4$	p=0.055,$mu_1 = 0.945$,$mu_2 = 465.9$
54	M/M/4, $M/Hr_2/4$	$M/Hr_2/4$	p=0.036,$mu_1 = 0.964$,$mu_2 = 239.3$
63	M/M/4, $M/Hr_2/4$	$M/Hr_2/4$	p=0.064,$mu_1 = 0.936$,$mu_2 = 453.0$
72	M/M/4, $M/Hr_2/4$	$M/Hr_2/4$	p=0.060,$mu_1 = 0.940$,$mu_2 = 419.5$
86	M/M/4, $M/Hr_2/4$	$M/Hr_2/4$	p=0.062,$mu_1 = 0.938$,$mu_2 = 452.0$
100	M/M/4, $M/Hr_2/4$	$M/Hr_2/4$	p=0.065,$mu_1 = 0.935$,$mu_2 = 510.8$
119	M/M/4, $M/Hr_2/4$	$M/Hr_2/4$	p=0.040,$mu_1 = 0.960$,$mu_2 = 231.0$

TABLE 7.1 – Résultats de l'identification de modèle de la boîte base de données

L'injection de charge auto-régulée s'est faite sur 14 paliers de charge succéssifs, pour atteindre 119 utilisateurs virtuels et l'exécution a duré 5 minutes. La ressource qui a atteint le seuil de saturation défini en premier est le processeur (arrêt à 82% d'utilisation).

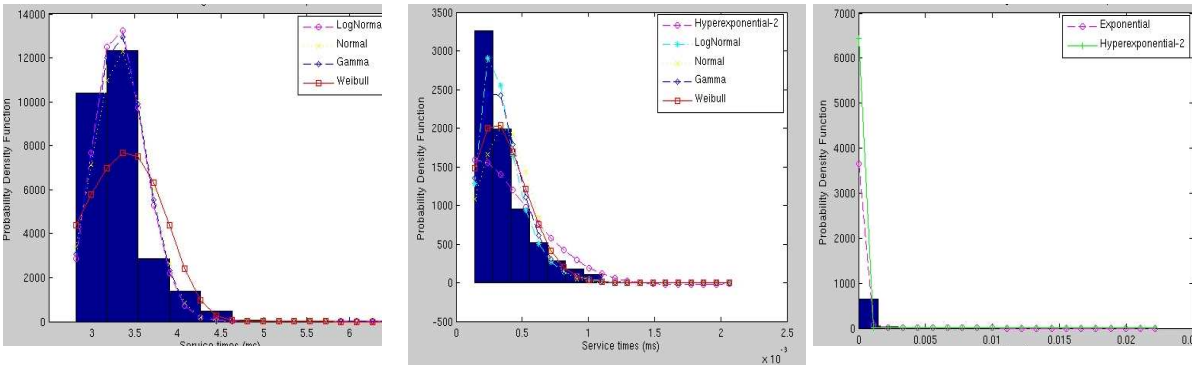

FIGURE 7.6 – Analyse de l'échantillon de la boîte Base de données : à charge faible(gauche), moyenne (milieu) et forte (droite)

L'outil d'identification du modèle récupère les mesures et détermine l'échantillon du temps de services associé à chaque palier. Chaque échantillon est analysé en utilisant les tests d'ajustement statistiques et la méthode graphique. Diverses distributions ont été identifiées avec leurs paramètres, notamment l'exponentielle, l'hyper-exponentielle à deux étages, la Log-normale, la gamma et la Weibull. Le tableau 7.1 décrit les modèles candidats identifiés de cette expérimentation.

Pour chaque niveau de charge, on choisit le modèle le plus approprié (donnée en caractères gras). Selon les tests d'ajustement statistiques, les meilleurs modèles sont ceux qui ont une meilleure p-valeur. La colonne « paramètres » indique les paramètres estimés pour le modèle selectionné.

Charge	Modèles identifiés	Modèle sélectionné	Paramètres
1	$M/\Gamma/3$, M/LN/3, M/Norm/3, M/Wbl/3	M/LN/3	m=-4.28, sigma=0.13
10	$M/\Gamma/3$, M/LN/3, M/Norm/3, M/Wbl/3	M/LN/3	m=-4.45, sigma=0.16
19	$M/\Gamma/3$, M/LN/3, M/Norm/3, M/Wbl/3	M/LN/3	m=-4.54, sigma=0.20
28	$M/\Gamma/3$, M/LN/3, M/Norm/3, M/Wbl/3	M/LN/3	m=-4.60, sigma=0.18
37	$M/\Gamma/3$, M/LN/3, M/Norm/3, M/Wbl/3	M/LN/3	m=-4.65, sigma=0.24
46	$M/\Gamma/3$, M/LN/3, M/Norm/3, M/Wbl/3	M/LN/3	m=-4.70, sigma=0.26
55	$M/\Gamma/3$, M/LN/3, M/Norm/3, M/Wbl/3	$M/\Gamma/3$	a=15.82, b=0.001
64	$M/\Gamma/3$, M/LN/3, M/Norm/3, M/Wbl/3	M/LN/3	m=-4.70, 0.30, sigma=0.47
82	$M/\Gamma/3$, M/LN/3, M/Norm/3, M/Wbl/3	M/LN/3	m=-4.73, sigma=0.36
100	$M/Hr_2/3$, $M/\Gamma/3$, M/LN/3, M/Norm/3, M/Wbl/3	M/LN/3	m=-4.81, sigma=0.37
118	$M/Hr_2/3$, $M/\Gamma/3$, M/LN/3, M/Norm/3, M/Wbl/3	M/LN/3	m=-4.79, sigma=0.38
136	M/M/3, $M/Hr_2/3$, M/Norm/3	$M/Hr_2/3$	p=0.13, $mu_1 = 0.86$, $mu_2 = 77.65$
162	M/M/3, $M/Hr_2/3$	$M/Hr_2/3$	p=0.09, $mu_1 = 0.90$, $mu_2 = 27.25$

TABLE 7.2 – Résultats de l'identification de modèle de la boîte EJB

Pour les petites et moyennes charges (de 1 à 36 utilisateurs virtuels), le modèle sélectionné est le modèle M/LN/4, car les tests statistiques donnent la plus grande p-valeur pour la distribution lognormale. Cette distribution peut être expliquée par le fait que les temps de service augmentent avec la charge et montre une dégradation du service de la boîte noire au cours des périodes de fort accroissement de charge.

Pour des charges plus élevées, le modèle $M/Hr_2/4$ est choisi pour la même raison. La figure 7.6 montre les histogrammes des temps de service avec les distributions identifiées.

7.4.5 Modèle du conteneur EJB

L'expérimentation sur le tiers EJB est réalisée sur un serveur Linux avec deux processeurs Xeon 2 GHz, et 1 Go de RAM. Nous avons déployé les mêmes injecteurs que ceux utilisés pour la base de données. L'injection de charge auto-régulée s'est faite sur 12 paliers pour atteindre 162 utilisateurs virtuels en 8 minutes. La figure 7.7 montre le profil de charge résultant. La ressource saturée en premier, selon les seuils définis, le processus, avec 86% d'utilisation.

En suivant les mêmes procédures d'expérimentation et d'analyse, on obtient les modèles présentés dans le tableau 7.2. Les distributions de temps de service identifiées sont l'exponentielle, l'hyper-exponentielle, la Log-normale, la gamma et la Weibull. En utilisant le même critère de sélection, le modèle M/Ln/3 est retenu pour tous les niveaux de charge, sauf pour les deux derniers. Comme dans le cas de la base de données, cela peut être expliqué par l'apparition des temps de service plus lents. Le système se comporte alors comme une file d'attente à deux serveurs avec des vitesses de service différents ($\mu_1 = 0,86$ $\mu_2 = 77,65$). Ce changement de comportement est expliqué par le rapprochement de la zone de saturation. Ce phénomène est identique à celui observé au niveau du tiers base de données, mais, il apparait largement après (136 utilisateurs virtuels/s pour l'EJB contre 45 utilisateurs virtuels/s pour la BD) cela annonce que le premier du goulet d'étranglement sera observé sur le tiers base de données (voir section 8.11)

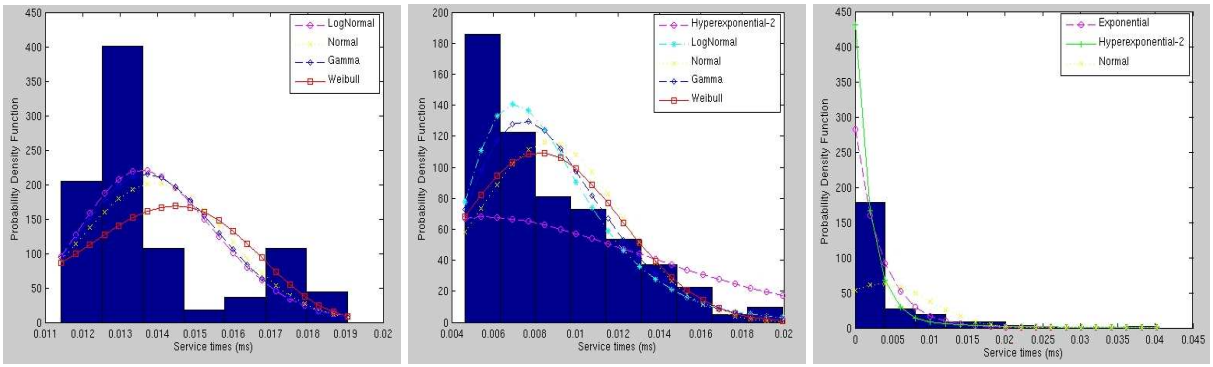

FIGURE 7.7 – Analyse de l'echantillon de la boîte EJB : à faible (gauche), moyenne (milieu) et forte (droite) charge

7.4.6 Modèle du conteneur Web

L'expérimentation sur le tiers serveur Web est réalisé sur un serveur Linux avec deux processeurs PIII à 1.2 GHz, et 1 Go de RAM. Nous avons déployé les mêmes injecteurs et l'injection de charge auto-régulée s'est faite en 6 étapes pour atteindre 16 utilisateurs virtuels en 7 minutes et 48 secondes. La figure 7.8 montre le profil de charge résultant. La ressource saturée en premier est la mémoire tas de la JVM, qui est passée en dessous du seuil de mémoire libre.

Charge	Modèles identifiés	Modèle sélectionné	Paramètres
1	$M/\Gamma/1$, M/LN/1, M/Norm/1, M/Wbl/1	$M/\Gamma/1$	a=76.87, b=0.001
4	$M/\Gamma/1$, M/LN/1, M/Norm/1, M/Wbl/1	M/LN/1	m=-3.25, sigma=0.13
7	$M/\Gamma/1$, M/LN/1, M/Norm/1, M/Wbl/1	$M/\Gamma/1$	a=62.92, b=0.001
10	$M/\Gamma/1$, M/LN/1, M/Norm/1, M/Wbl/1	M/LN/1	m=-3.49, sigma=0.14
13	$M/\Gamma/1$, M/LN/1, M/Norm/1, M/Wbl/1	$M/\Gamma/1$	a=36.82, b=0.001
16	$M/\Gamma/1$, M/LN/1, M/Norm/1, M/Wbl/1	M/LN/1	m=-3.67, sigma=0.16

TABLE 7.3 – Résultats de l'identification de modèle de la boîte Web

Les modèles obtenus sont décrits dans le tableau 7.3. Les distributions de temps de service identifiées sont la Log-normale, la normal, la gamma et la Weibull. Les modèles sélectionnées sont simultanément M/Ln/1 et $M/\Gamma/3$.

7.4.7 Choix et validation du modèle global

On crée un modèle global du réseau de files d'attente représentant l'application *SampleCluster* pour un niveau de charge donné, par exemple, 16,2 requêtes/s. On choisit le modèle $M/Hr_2/4$ pour la base de données, le modèle $M/Hr_2/3$ pour le conteneur EJB et $M/Ln/1$ pour le conteneur web. Puis, on compare le temps de réponse moyen estimé à partir du réseau aux mesures réelles obtenues au même niveau de charge. Le tableau 7.4 montre les valeurs

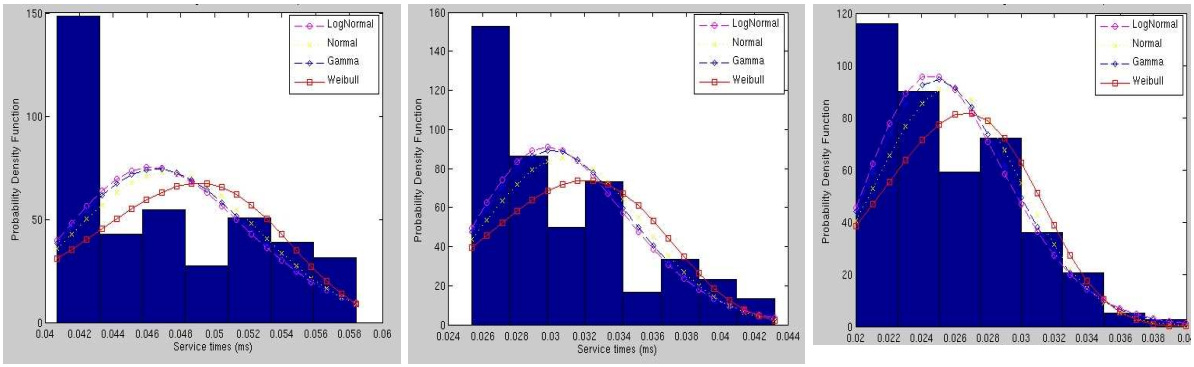

FIGURE 7.8 – Analyse de l'échantillon de la boîte Web : à charge faible (gauche), moyenne (milieu) ou forte(droite)

Indice de performance	Estimation avec un modèle composé	Mesures
Charge	16.2 requêtes/s	16.2 requêtes/s
Temps de réponse	52 ms	52 ms

TABLE 7.4 – Comparaison entre les indices de performance théoriques et empiriques

moyennes des indices de performance théoriques calculés et mesurés au niveau de charge de 16,2 requêtes/s. L'égalité entre les mesures de temps de réponses et les valeurs empiriques (52 ms) représente une validation partielle de notre processus de modélisation automatique par rapport à l'application *SampleCluster* pour une charge fixe. Cette même validation doit être réalisée pour différents niveaux de charge avec différents modèles.

7.4.8 Efficacité de la décomposition

Indice de performance	Modèle composé	Modèle du système avec une seule boîte	Mesures
Charge	16.2 requêtes/s	16.2 requêtes/s	16.2 requêtes/s
Temps de réponse moyen	52 ms	51 ms	52 ms
Nombre moyen de client	0.87	0.80	-

TABLE 7.5 – Comparaison entre les indices de performance théoriques pour les modèles avec ou sans décomposition

Une autre question qui peut être posée à ce niveau concerne l'impact de la décomposition sur la précision du modèle. Pour vérifier cet impact, on applique l'injection de charge automatique et le processus d'identification de modèle sur toute l'application *SampleCluster* en la considérant comme une seule boîte noire. Puis, on analyse les performances du modèle obtenu à un niveau de charge donné et on compare les résultats du réseau de files d'attente avec les mesures réelles. L'objectif ici est de vérifier et de comparer la précision des modèles avec ou sans décomposition. Le tableau 7.5 montre que la précision de la modélisation avec décomposition

en 3 boîtes noires est légèrement meilleure à celle du modèle sans décomposition pour la charge étudiée. En effet, le temps de réponse moyen estimé à partir d'un modèle global avec une seule boîte noire est de 51 ms au lieu de 52 ms pour le modèle. Cette validation partielle montre que le niveau de décomposition choisi permet d'obtenir un modèle plus précis que le modèle sans décomposition. Néanmoins, cette question n'est qu'une ouverture sur une problématique plus large qui pourrait être étudiée d'une manière plus exhaustive : celle du rapport entre le niveau de décomposition et la précision du modèle.

Nous justifions avant tout la décomposition par le besoin de détection des goulets d'étranglement, ainsi que et de reconfiguration qui sera étudiée dans la section 7.6.

7.5 Du modèle vers le dimensionnement

Après avoir choisi et validé un modèle global, l'étape de prédiction des performances permet de construire un nouveau jeu de paramètres optimisant les performances du système. Cette étape finale consiste à résoudre numériquement ou par simulation le modèle de réseau de files d'attente associé. Le choix de la méthode de résolution dépend de la nature de réseau de files d'attente (voir les figures 2.5 et 2.9), de la précision des estimations et du temps de résolution.

Contrainte de réactivité du dimensionnement : la phase de dimensionnement est souvent contrainte par le temps. En cas de panne ou de blocage d'un système en cours d'exécution, la détection et la correction de du problème doit se faire dans les plus brefs délais afin d'éviter les conséquences néfastes comme décrite dans l'introduction de cette thèse. C'est pourquoi il est important de réduire au maximum le temps de résolution du modèle de réseau de file d'attente, étape la plus coûteuse de notre processus de dimensionnement. En effet, l'étape de modélisation se réduit à la composition des modèles de boîte noire «prêt à l'emploi» disponibles dans le dépôt de modèles (voir figure 4.1). La figure 7.9 illustre le fait que plus le modèle global est détaillé et précis plus le temps de résolution est grand. Un compromis entre temps de résolution ou de simulation, et précision du modèle construit doit être pris en considération dans le choix de ce modèle afin de réduire le temps total de la phase de dimensionnement. Le tableau 2.1 met en évidence la complexité en temps d'exécution et en consommation mémoire des algorithmes de résolution de réseau de files d'attente à forme produit. Un tel comparatif permet à l'expérimentateur de choisir la technique de résolution en fonction de sa rapidité et de sa précision, selon les exigences et les possibilités du contexte de la prédiction.

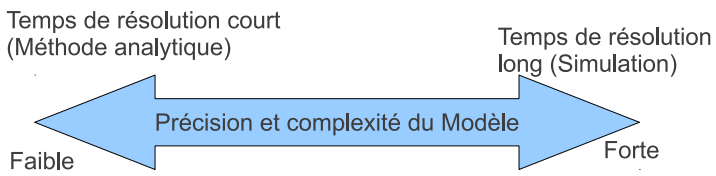

FIGURE 7.9 – Temps de résolution et précision du modèle global

Formulation du problème de dimensionnement : l'expérimentateur doit définir les ob-

jectifs du dimensionnement (SLO) et déterminer les paramètres de performance à estimer par cette étape de résolution. Dans la majorité des cas, ces objectifs se résument à réduire le temps de réponse de bout en bout afin de répondre rapidement aux requêtes des clients et de servir le maximum de clients en un minimum de temps. Néanmoins, d'autres objectifs peuvent guider le dimensionnement, tels que la réduction d'un taux de rejet, la minimisation des ressources utilisées ou tout simplement l'évitement d'un goulet d'étranglement.

Par exemple, dans l'application *SampleCluster*, l'augmentation de 10% de la charge en entrée nécessite-t-elle l'ajout de ressource afin de garder un temps de réponse de bout en bout inférieure à 0,1s ? Pour répondre à cette question, il convient de déterminer l'intervalle de charge associé à l'augmentation souhaitée et de produire le modèle global. Pour la charge initiale de 16 requêtes/s, la nouvelle charge à tester est de 16+16*10/100 = 17,6 requêtes/s. D'après les tableaux des modèles identifiés pour *SampleCluster* le modèle choisi restera le même pour cette nouvelle charge. Il ne reste plus qu'à simuler le réseau de file d'attente pour cette nouvelle charge. Le temps de réponse moyen estimé est de 64 ms, ce qui ne viole pas le SLO. Ainsi, il n'est pas nécessaire d'ajouter des ressources si on augmente la charge en entrée de 10%.

Supposons maintenant que cette simulation montre le contraire c'est-à-dire que cette augmentation de charge engendre la violation du SLO. Il devient alors nécessaire de rechercher la ou les nouvelles configurations permettant d'assumer une telle charge tout en respectant le SLO. Cela consiste à détecter en premier lieu, le tiers responsable de cette violation du SLO, et d'agir en second lieu de la sorte à augmenter les ressources associées à ce tiers afin de ramener le système à la qualité de service souhaitée. Ce problème n'est autre qu'un problème de détection et de suppression des goulets d'étranglement que nous traiterons dans la section suivante.

7.6 Détection et suppression des goulets d'étranglement

Indice de Performance	Solution 1	Solution 2	Solution 3
Temps de réponse	4186 sec	1855 ms	1454 ms
Débit	70.88 requests/s	106.80 requests/s	117.20 requests/s
Utilisation Web 1	0.90	1	1
Utilisation Web 2	0.90	1	1
Utilisation Web 3	-	1	1
Utilisation EJB 1	0.70	1	0.59
Utilisation EJB 2	-	-	0.60
Utilisation MySQL	0.09	0.14	0.15

TABLE 7.6 – Tableau récapitulatif des indices de performance pour différentes configurations

La détection des goulets d'étranglement consiste à surveiller l'évolution du taux d'utilisation des ressources (processeur, mémoire, réseau, etc.) afin de réagir avec dégradations de niveau de service. Il esr nécessaire de trouver la boîte noire ou la partie du système (ressources, partie

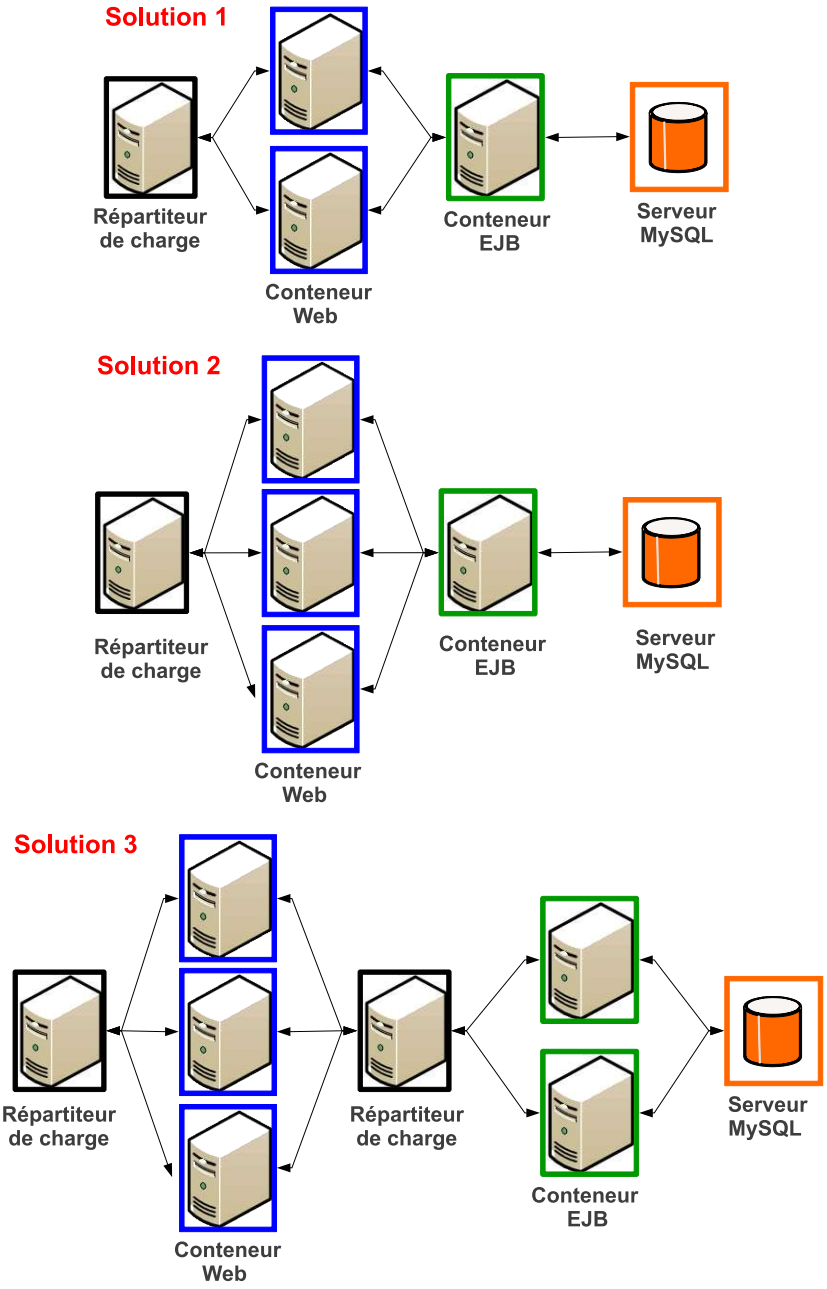

FIGURE 7.10 – Plusieurs configurations envisagées pour éviter un goulet d'étranglement

d'applicatif) responsable de ces dégradations. Nous montrons comment notre approche permet de déterminer la « meilleure configuration » du système à appliquer chaque fois qu'un goulot d'étranglement empêche le système d'atteindre le SLO. Bien entendu, la notion de meilleure configuration est une question de point de vue. Du point de vue client, seuls les critères de qualité de service compte, tels que les temps de réponse, la disponibilité du service et de fiabilité. Du point de vue du gestionnaire du système, un compromis doit être trouvé entre les dépenses de fonctionnement, d'une part et la satisfaction client, d'autre part. Pour notre approche, on se limite à évaluer plusieurs configurations et à déterminer la configuration qui garantie la meilleure qualité de service sans se soucier des coûts. Dans le cas où plusieurs configurations seront validées par notre processus de prédiction, le gestionnaire pourra choisir une configuration en prenant en considération la fonction coût.

La procédure de détection du goulet d'étranglement est lancée chaque fois qu'on détecte une violation du SLO. Le goulet d'étranglement dépend en premier de la charge d'entrée de chaque boîte. Dans le cas d'un système existant, on mesure cette charge puis on mesure les taux d'utilisation des ressources et les temps de réponse au niveau de chaque boîte. Dans le cas d'une reconfiguration d'optimisation à tester, on estime la charge en entrée en fonction de la charge du réseau de file d'attente et de la matrice de routage. Puis en se basant sur les mesures des expérimentations initiales des boites noires en isolation associée à cette charge, on détermine l'ordre d'exploration des boîtes considérées comme des éventuelles goulets d'étranglement.

Pour une charge fixée, on mesure/estime les taux d'utilisation des ressources et les temps de réponse au niveau de chaque boîte. La boîte qui a une utilisation maximale est considéré comme la boîte responsable du goulet d'étranglement. Malgré que l'utilisation des ressources (cpu, mémoire) représentent des bons estimateurs de l'utilisation théorique, il s'avère que cette hypothèse n'est pas toujours vrai. En effet, il se peut qu'une boîte admette une utilisation CPU maximale ne soit pas la cause du goulet d'étranglement. Dans ce cas, les boîtes seront testées une à une dans l'ordre d'exploration défini.

La détection et l'évitement d'un goulet d'étranglement ne pourront être affirmé qu'après avoir évalué les performances globales de la reconfiguration optimisant le tiers considéré comme goulet d'étranglement et avoir une amélioration considérable de qualité de service.

La figure 5.7 décrit la phase de dimensionnement de notre démarche comme une boucle avec un feedback permettant de tester d'une manière itérative plusieurs configurations. Ces configurations sont des solutions d'optimisation permettant l'ajustement de l'infrastructure afin d'éviter les goulets d'étranglement. Ces ajustements varient en fonction de la distance entre les performances estimées et les performances souhaitées et peuvent aller d'une simple reconfiguration de la mémoire allouée jusqu'à la duplication du serveur Web ou du serveur de base de donnée.

Dans le cas de notre application *SampleCluster*, on fixe la charge d'entrée à 180 requêtes/s, l'étude des boîtes en isolation et leurs utilisations permettent d'établir l'ordre d'exploration des boîtes noires pour la recherche des goulets d'étranglement. Nous commençons alors à tester dans l'ordre les reconfigurations liées au conteneur Web, au conteneur EJB puis à la base de donnée. La simulation de la configuration encours donne une forte utilisation pour les boîtes

noires "conteneur Web" et "conteneur EJB" et qui atteint 1 pour le "conteneur Web". le temps de réponse de bout en bout pour cette configuration est estimé à $9778s$ (2.71 heures) ce qui représente évidemment une qualité de service inacceptable et montre clairement l'existence d'un goulet d'étranglement.

Le problème est donc de déterminer le ou les tiers responsables de ce goulet d'étrangement et de prévenir le système en cas de déplacement de ce goulet (problème décrit dans la section 4.4.2). Étant donné que notre approche repose sur une méthode itérative de test de configuration et sur l'évaluation globale du réseau de file d'attente, tout problème de déplacement du goulet d'étranglement sera résolu itérativement. En effet, dans notre cas d'optimisation du serveur web et son éventuelle reconfiguration permettra de déplacer le goulet d'étranglement du tiers web au tiers EJB. Cette dernière solution ne permet qu'une résolution locale du goulet d'étranglement (tiers web). Vu qu'on se base sur une évaluation complète du modèle global, notre processus itératif ne s'arrêtera pas tant que le SLO n'est pas encore satisfait et finira par choisir la meilleure configuration parmi les solutions testées.

La figure 7.10, montre trois solutions de reconfiguration à tester itérativement pour l'application *SampleCluster*. La première solution propose de dupliquer le conteneur web sur 2 serveurs, la deuxième propose d'utiliser trois serveurs pour ce dernier et la troisième duplique 3 fois le serveur web et 2 fois le tiers EJB. Les indices de performances et d'utilisation des ressources de chaque solution sont regroupés dans le tableau 7.6. Ces résultats estimés par notre processus de prédiction de performance montrent que la 2ième et la 3ième solution représentent des configurations possibles permettant de satisfaire le SLO avec une réduction de temps de réponse de bout en bout qui passe de 4186 secondes à 1,4 seconde et augmente le débit global de l'application qui passe de 70 requêtes/seconde à 117requêtes/seconde. Malgré ces résultats satisfaisants qui permet une grande amélioration de la qualité de service et de la charge supportée par le système, ces solutions ne résoudent pas le problème d'utilisation du tiers Web (cas de la solution 2 et 3) et d'utilisation de EJB (cas de solution 2) qui restent à 1. Etant donnée ces résultats, l'administrateur pourra retenir une des solutions 2 ou 3 en prenant en considération le coût d'investissement et toute en restant vigilant sur le comportement de tiers EJB et Web. Il pourra aussi chercher une solution plus prudente en relançant le processus de prédiction des performances avec des nouvelles configurations permettant de réduire l'utilisation du serveur Web. La "meilleur configuration" conseillée par notre processus de prédicition est la solution 3 bien qu'elle ne résout pas l'utilisation du tiers Web. En effet, l'utilisation du tiers web est calculé par rapport à l'utilisation de la ressources mémoire virtuelle qui ne représente pas un problème majeur pour l'exécution de l'application et qui pourra être expliqué par une périodicité de passage de garbage collector assez longue.

7.7 Conclusion

Dans ce chapitre, nous avons présenté notre méthode de composition du modèle global et sa validation. Une fois ce modèle validé, nous avons montré comment choisir la méthode de résolution/simulation permettant de dimensionner le système sous test.

Toute cette démarche a été illustrée et validée par l'application *SampleCluster*. Ceci a montré en premier la qualité du modèle global généré puis la simulation qui a permis de dimensionner l'application. Finalement, nous avons mis en évidence notre méthodologie de détection et d'évitement des goulets d'étranglement. Les goulet ont été identifiés en premier au niveau de la boîte noire web puis nous avons détecté son déplacement au niveau du tiers EJB.

Le processus itératif et réactif de test de configuration a permis de choisir une meilleure reconfiguration parmi plusieurs permettant d'éviter la dégradation de la qualité de service et de respecter le SLO.

L'application utilisée est une application J2EE. Néanmoins, toute notre démarche est générique, elle peut être appliquée pour toute application capable d'être décomposée, injectée puis tracée afin de générer les modèles des boîtes noires associés.

Le prochain chapitre présentera notre Framework de modélisation automatique "FAMI" ainsi que les outils de simulation et de résolution des modèles de file d'attente pour le dimensionnement.

Chapitre 8

Vers une infrastructure de dimensionnement automatique

8.1 Introduction

Les chapitres précédents ont présenté et validé notre approche de modélisation automatique et de dimensionnement. L'objectif de ce chapitre est de présenter notre framework implémentant cette approche. On commence par exposer les problématiques et les contraintes techniques de l'automatisation, de la modélisation et du dimensionnement. Puis, on présente les choix des infrastructures et des outils utilisés qui ont permis de répondre à ces besoins tout en étant générique et extensible et en assurant une qualité d'expérimentation. Nous présentons ensuite l'architecture de notre framework. A la fin de ce chapitre, une évaluation de cette architecture est proposée, ainsi que des améliorations possibles.

8.2 Besoins techniques et fonctionnels

Notre infrastructure d'évaluation est composée d'un ensemble de modules logiciels développés de manière autonome. Les principaux besoins techniques et fonctionnels sont :

- **Généricité :** le framework doit être générique afin de pouvoir expérimenter et de dimensionner une large classe de systèmes, à savoir les systèmes multi-tiers, Machine to Machine, etc. D'une part, pour l'expérimentation, on doit avoir la possibilité d'injecter et de sonder tous type de composants formant le système qu'on veut modéliser. D'autre part, pour la partie modélisation et résolution on doit pouvoir identifier des modèles divers et variés de file d'attente avec des lois de services markoviens, des lois normales et même des lois à queue lourde. Enfin, on doit avoir la possibilité de résoudre ou simuler tous les réseaux de files d'attente formés.
- **Modularité :** chaque entité fonctionnelle (identification du modèle, recherche du goulet

d'étranglement, calcul de temps de stabilisation) doit être isolée dans un module permettant son remplacement. Cela est intéressant lorsque on veut remplacer le module de recherche de goulet d'étranglement par un deuxième module implémentant une approche de recherche différente.
- **Efficacité :** un des objectifs premiers de l'automatisation est de réduire le temps global des expérimentations. Cette réduction concerne, par conséquent, toutes les étapes de notre expérimentation automatique, et particulièrement le calcul du temps de stabilisation et l'estimation du modèle qui peuvent dans des cas de modèles de files d'attente particuliers prendre des temps importants ce qui pourrait ralentir considérablement l'expérimentation.
- **Extensibilité :** le framework doit être extensible pour prendre en considération des nouveaux systèmes sous test à expérimenter et à dimensionner. Cette extensibilité se manifeste au niveau :
  - des injecteurs qui doivent générer des nouveau types de requêtes ou des nouveaux protocoles de communication,
  - des modèles qui doivent prendre en considération d'autres familles de lois statistiques et leurs tests d'hypothèse associés,
  - des algorithmes de résolution ou de simulation pour avoir des résultats ou des estimations plus précis en fonction du réseau de file d'attente.

Du point de vue fonctionnel, l'environnement propose :

- observation et mesure,
- traitement des données : synthèse,
- inférence et prédiction de performance.

Pour mieux comprendre ces fonctions, nous allons détailler chacune d'entre elles.

8.2.1 Observations et mesures

La fonction d'observation et de mesure correspond à l'étape de collecte des mesures dans notre expérimentation automatique. Cette fonction conditionne la qualité de tout le processus de modélisation et de dimensionnement. Deux méthodes de collecte sont possibles dans ce cas :

- soit une collecte de mesures dite passive, qui se base sur les traces d'exécution des systèmes sous test ou des boîtes noires en cours d'exécution normale ;
- soit une collecte de mesures active, qui se base sur une expérimentation dédiée.

Bien que la première solution puisse paraître plus attractive du fait qu'elle ne demande pas des expérimentations dédiées aux collectes des mesures, elle reste limitée du point de vue de la répartition des mesures par rapport à la charge en entrée. En particulier, étant donné qu'on ne contrôle pas la charge en entrée, comme nous l'avons vu dans la section 5.3, on ne pourra pas déterminer les estimateurs initiaux de la charge maximale et du taux de service, c'est pourquoi nous avons opté pour une phase de collecte active.

8.2.2 Contrôle de l'expérimentation autonome

Notre expérimentation se base donc sur une injection de charge autonome représentant une activité courante des experts de la mesure performances. Plusieurs outils existent [36, 39, 30] avec différents avantages et inconvénients. Le tableau 8.1 récapitule ces avantages et inconvénients pour différents outils selon plusieurs critères.

- Le premier critère est à la disponibilité et la modularité du code (open source ou non) pour la réutilisation du code et son extensibilité comme présenté au début de cette section.
- Le deuxième critère est le passage à l'échelle. Puisqu'on cherche à déterminer le comportement du système sous test jusqu'à sa charge maximale admissible. Les injecteurs de charge risquent eux-mêmes d'atteindre leurs limites.
- Le troisième critère est la possibilité d'avoir les mesures en temps expérimental réel. En effet, la montée en charge utilise les mesures du palier courant pour décider du palier suivant. Par conséquent, les mesures doivent être envoyées et traitées simultanément pour la prise de décision, sans toutefois perturber le fonctionnement du système d'injection (réseau complet) ce qui est a priori délicat à réaliser.
- Le dernier critère est la possibilité de modifier la charge à chaud afin d'imposer le niveau de charge du nouveau palier en cours d'expérimentation.

	HP LoadRunner	JMeter	Clif
Open source (extensibilité)	-	+	++
Passage à l'échelle	++	-	++
Mesures durant l'expérimentation	+	+	+
Modification de la charge à chaud	-	-	+
Sondes	++	-	+

TABLE 8.1 – Tableau de comparaison des injecteurs de charge

La comparaison des outils *HP LoadRunner*, *JMeter* et *Clif* montre que *HP loadRunner* passe mieux à l'échelle par rapport à la quantité de la charge injectée, ce qui représente un avantage considérable. *LoadRunner* est une solution propriétaire limitant la possibilité d'implémenter notre méthodologie expérimentale. *JMeter* est une solution open source qui peut être utilisée à ces fins, mais son principal inconvénient est sa limite de son passage à l'échelle. Ce récapitulatif montre bien l'avantage du framework Clif[31] malgré son implémentation en Java qui limite les performances à cause de *garbage collector*. CLIF se caractérise aussi par :

- un système de plugins permettant différents types d'injection et de mesures.
- un moteur d'injection de charge ISAC[32] avec des profils de charge et des scénario d'injection paramétrables.
- Une implémentation à base de composants qui permet plus de flexibilité à notre framework.

Cette solution a été retenue et a servi comme environnement de base pour notre framework.

8.2.3 Traitement statistique des données

Notre identification de modèle nécessite des mesures stables et avec un bruit minimal. Cela demande alors une utilisation d'outils statistiques classiques tels que (moyenne, médiane, écart type) pour la phase d'épuration des mesures, et une implémentation optimisée des algorithmes de test d'hypothèse. Deux choix sont possibles :

1. le premier est la ré-implémentation des algorithmes statistiques au sein de notre framework qui représente l'avantage d'accès rapide à ces implémentations et par conséquent un gain au niveau du temps de calcul. Le second est l'utilisation d'un outil statistique implémentant ces tests d'hypothèses tel que Matlab[35], Scilab[97] ou R[41, 54].
2. Cette deuxième solution est de se décharger de l'implémentation en utilisant un outil intégrable à notre framework.

Scilab est un logiciel open source de traitement de signal, de modélisation et de simulation. Il implemente aussi plusieurs fonctions d'analyse statistique mais il reste limité par rapport à R. Ce dernier a l'avantage d'être un logiciel libre, spécialement conçu pour l'analyse statistique. Il regroupe une large communauté scientifique active (beaucoup d'optimisations et de correctifs des algorithmes statistiques sont disponibles). Matlab est un outil plus complet, qui implémente plusieurs variantes de test d'hypothèse. Son principal inconvénient est d'être propriétaire. R a été retenu pour cette tâche pour plusieurs raisons :

– Premièrement, R est totalement intégrable à notre framwork grâce à son API d'intégration R-java.
– Deuxièmement, le temps total d'appel des algorithmes R à travers son API reste comparable au temps total avec une implémentation interne lorsqu'on travaille avec des petits échantillons, ce qui est notre cas. La taille de nos échantillons par palier varie entre 30 et 200 mesures. [1]

L'utilisation de R nous a permis d'utiliser plusieurs tests d'hypothèse pour les mêmes mesures afin de s'assurer de la validité des résultats en l'absence des contraintes de temps/durée (cas d'une qualification d'une boîte en phase préparatoire).

Notons que nous avons utilisé Matlab dans une des versions de notre framework afin d'utiliser le test d'hypothèse d'Anderson Darling.

8.2.4 Inférence et dimensionnement

La résolution du modèle de réseau de file d'attente associé à une configuration est cruciale. En effet, le rejet d'une configuration fiable et économique à cause d'une imprécision d'un algorithme, expose l'administrateur à des pertes en temps d'expérimentation et à des pertes économiques.

1. On rappelle que la validité des tests d'hypothèse dépend de la taille de l'échantillon. cette taille est comprise entre 30 et 200 lorsqu'on travaille avec les tests de kolmogorov-smirnov ou du χ^2

Du point de vue fonctionnel, l'outil doit fournir une variété d'estimateurs des indices de performances avec leur intervalle de confiance (temps de réponse, utilisation et débit). L'intervalle de confiance est une donnée nécessaire pour quantifier la précision de nos résultats. L'outil doit aussi fournir des modèles statistiques variés pour modéliser les inter-arrivés et les lois de service de chaque file d'attente avec précision. Citons par exemple la famille exponentielle (hyper-exponentielle, hypo-exponentielle, Erlang) ou la famille des lois normales (normale, lognormal). Enfin, l'outil doit supporter divers algorithmes d'analyses exacts et estimés pour des modèles de files d'attentes markoviens ou généraux, à un ou plusieurs serveurs, tels que les algorithmes décrit dans le chapitre 2. Par ailleurs, l'outil doit satisfaire les différents besoins non fonctionnels décrits au début de la section et doit être intégrable et libre.

Java Modelling Tools JMT [13, 16] est une suite, en terme d'outils développés au sein du laboratoire d'évaluation de performance de l'université Polytechnique de Milan. Il comprend en particulier un simulateur JSIM[15] pour des réseaux de file d'attente à forme non produit et un solveur JMVA[14] implémentant l'algorithme MVA. L'outil QNAP2 [86] est un outil du *framework modline* commercialisé par Astek (ex-Simulog). QNAP2 implémente un nombre important de solutions analytiques (convolution, BCMP, MVA, diffusion, MARCA) et un simulateur avec une variété de distributions statistiques. Il permet aussi la définition de classes de clients avec différentes politiques d'ordonnancement. Il permet ainsi la résolution et l'estimation de réseaux de serveurs et de files d'attente très généraux. JavaSim[73] est un outil open source de simulation permettant une simulation à événements discrets. La librairie dispose de plusieurs distributions des familles de lois exponentielles et normales.

Le tableau 8.2 compare ces différents outils par rapport aux besoins fonctionnels et non fonctionnels. Malgré l'avantage de QNAP2 par rapport aux besoins fonctionnels, la non disponibilité de son code constitue une contrainte forte pour l'extensibilité de notre outil. Nous avons choisi JMT qui répond au mieux aux besoins cités précédemment et qui pourra être enrichi facilement par d'autres algorithmes d'analyse plus généraux. De plus, JMT est facilement intégrable à notre framework.

	QNAP2	JavaSim	JMT
Solution analytique	++	-	+
Simulation	++	+	+
Open Source	-	+	+
Extensibilité	-	+	+

TABLE 8.2 – Tableau de comparaison des solveurs de RFA

8.3 Architecture globale de l'environnement de modélisation et de dimensionnement

8.3.1 Architecture de l'environnement modélisation automatique

a Vue fonctionnelle

Pour expliquer les différentes parties de l'expérimentation automatique et de la modélisation, on se base sur une vue fonctionnelle de l'architecture. Cette vue fonctionnelle représentée par la figure 8.1 répartit les fonctionnalités de contrôle et de modélisation en composants logiques. On retrouve alors les principaux composants d'une plate-forme d'injection de charge auto-régulée (voir Figure 5.4), à savoir les injecteurs, les sondes, le contrôleur d'injection et le système sous test, auxquels on ajoute des composants de modélisation et de calcul. Dans la suite, nous détaillons chacun de ces composants et nous décrivons leurs interactions à travers un scénario d'expérimentation.

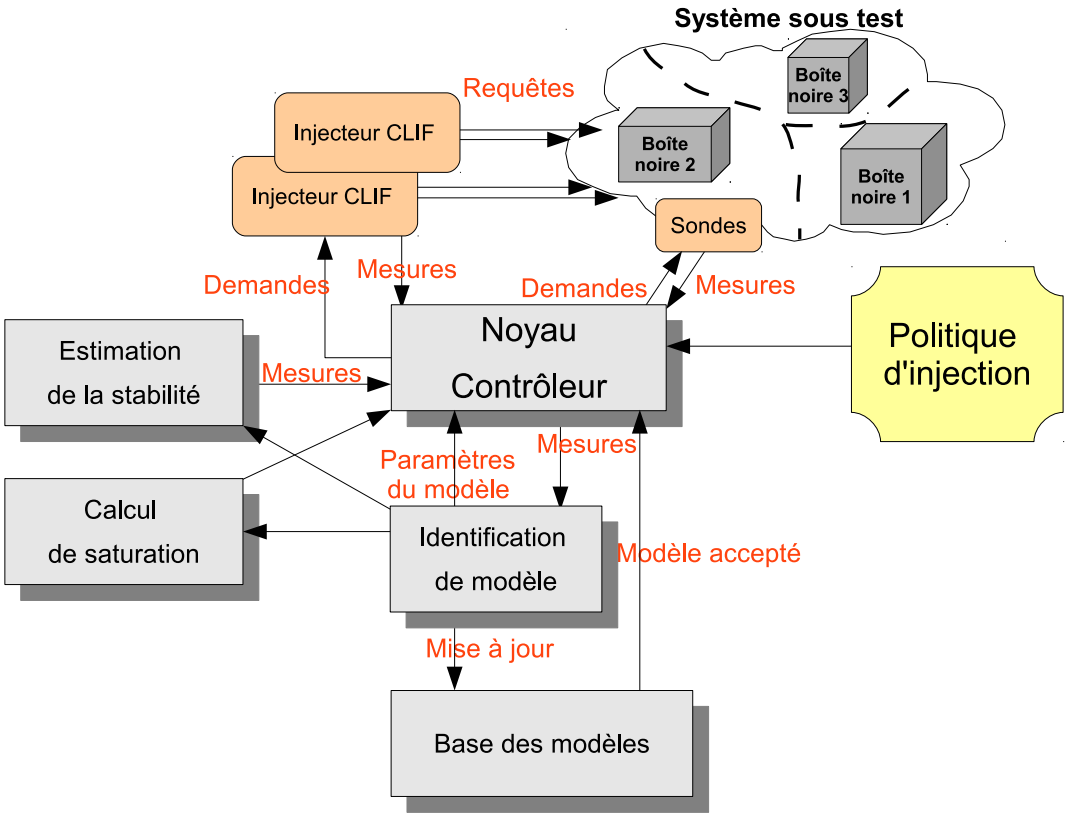

FIGURE 8.1 – Vue fonctionnelle de l'environnement d'expérimentation et de modélisation

Le noyau contrôleur : dans cette architecture, le composant logique «Noyau contrôleur» constitue le composant central. C'est la partie qui orchestre toute l'expérimentation automatique tout en assurant les communications nécessaires entre les différents modules :

- il fixe le niveau de charge généré par les injecteurs sur la boîte noire sous test en se basant sur la politique d'injection fournie ;
- il surveille les mesures que les sondes collectent et détermine celles qui sont stables ;
- il guide le processus d'identification de modèle de performances ;
- il surveille le bon fonctionnement de l'expérimentation et l'arrête avant saturation.

Toutes ces fonctions sont assurées par les composants dédiés suivant :

Calcul de saturation : en fonction des mesures collectées par les sondes (l'estimation de l'utilisation) et du modèle identifié, ce composant détermine et met à jour la charge maximale de bon fonctionnement de la boîte noire sous test conformément à la méthode décrite dans la section 5.3.5. Il assure ainsi le bon déroulement de l'expérimentation automatique en évitant la saturation et permet au «noyau contrôleur» d'arrêter l'expérimentation au bon moment.

Estimation de la stabilité : Ce composant calcule le temps de stabilisation théorique en se basant sur les modèles identifiés de la boîte se trouvant dans le composant «identification de modèle». Ce temps sera corrigé expérimentalement conformément à la procédure décrite à la section d. La détermination de cette durée détermine les zones de mesures stables nécessaires pour les composants d'identification de modèle et de contrôle.

Identification des modèles : ce composant récupère des échantillons de mesures stables du composant «noyau contrôleur» pour identifier les distributions d'inter-arrivée et de service en utilisant les tests d'hypothèse statistiques. Ensuite, ces modèles sont stockés dans le composant «base des modèles» et sont à leur tour fournis aux composants d'estimation de la stabilité et de la saturation.

Base des modèles : ce composant sert à stocker pour chaque niveau de charge les différents modèles de file d'attente validés. Il joue le rôle d'interface entre l'environnement de modélisation et l'environnement de dimensionnement.

Politique d'injection : ce composant exprime les divers besoins de l'expérimentateur pour le contrôle des tests en charge. C'est l'algorithme utilisateur permettant d'exprimer sa manière de contrôler l'expérience, les injecteurs, les conditions qu'il met pour intervenir, les ressources critiques qui dirigent le contrôle, etc. Pour donner à l'expérimentateur la possibilité d'exprimer la politique d'injection, nous avons opté pour un langage de description de la politique formé par un ensemble d'instructions d'initialisation qui sont exécutées au début de l'expérience et d'un ensemble d'instructions que le contrôleur exécute à chaque itération. Après l'exécution de ces instructions, le «noyau contrôleur» envoie les ordres pour modifier les paramètres des injecteurs et des sondes. Chaque instruction possède alors comme arguments le nom du paramètre à changer et la nouvelle valeur du paramètre. Pour que ces instructions soient suffisamment riches, on utilise quelques structures de contrôle comme des structures conditionnelles. Enfin, d'autres paramètres du profil de charge automatique sont fournis, à savoir la valeur courante de la charge maximale C_{max}, la durée d'injection et la durée échantillonnage (voir la section 5.3.4), pour mieux développer la politique d'injection de charge souhaitée.

Pour implémenter la politique d'injection, nous avons opté pour un fichier de configuration au format XML. Ce fichier est parcouru et exécuté par le composant «Noyau contrôleur». Chaque instruction est décrite par une balise XML qui englobe les balises des variables et des constantes. Ces différentes balises XML et les balises décrivant les structures de contrôle permettent d'exprimer les politiques d'injections les plus complexes. Les spécifications de ce fichier de contrôle sont explicitées dans l'annexe.

b Scénario d'exécution

Pour mieux comprendre les interactions entre ces différents composants, on présente ce scénario d'exécution :

1. Au début de l'expérience, le composant «Noyau contrôleur» reçoit la politique d'injection à exécuter.
2. Le «Noyau contrôleur» exécute l'étape d'initialisation (injection unitaire) qui sera spécifiée par la politique.
3. Les sondes répondent au «Noyau contrôleur» en envoyant les mesures.
4. Le noyau contrôleur achemine ces mesures au composant de calcul de saturation et au composant d'estimation de la stabilité afin de déterminer les mesures fiables et d'estimer une charge maximale initiale $Cmax_0$ en supposant avoir un unique serveur (K=1).
5. Le contrôleur exécute la boucle de contrôle. Pendant cette boucle, après chaque palier de charge :
   - le «Noyau contrôleur» récupère les mesures correspondant à ce palier, après des différentes sondes.
   - vérifie s'il y a saturation ou dépassement du SLO en consultant le composant de calcul de saturation auquel cas on sort de la boucle de contrôle. Sinon on récupère les nouveaux paramètres $Cmax$ et K.
   - Le composant d'« estimation de la stabilité» envoie le temps de stabilisation au « Noyau contrôleur».
   - Le « Noyau contrôleur» passe les mesures stables au composant d'« Identification du modèle».
   - Ce composant identifie et valide un ensemble de modèles de file d'attente et les envoie aux composants «Noyau contrôleur», «calcul de saturation» et «estimation de stabilisation».
   - Ces lois sont stockées dans le composant « base de modèle».
   - Suivant la politique d'injection, le «Noyau contrôleur» détermine la nouvelle charge à injecter au système et envoi l'ordre au injecteurs.

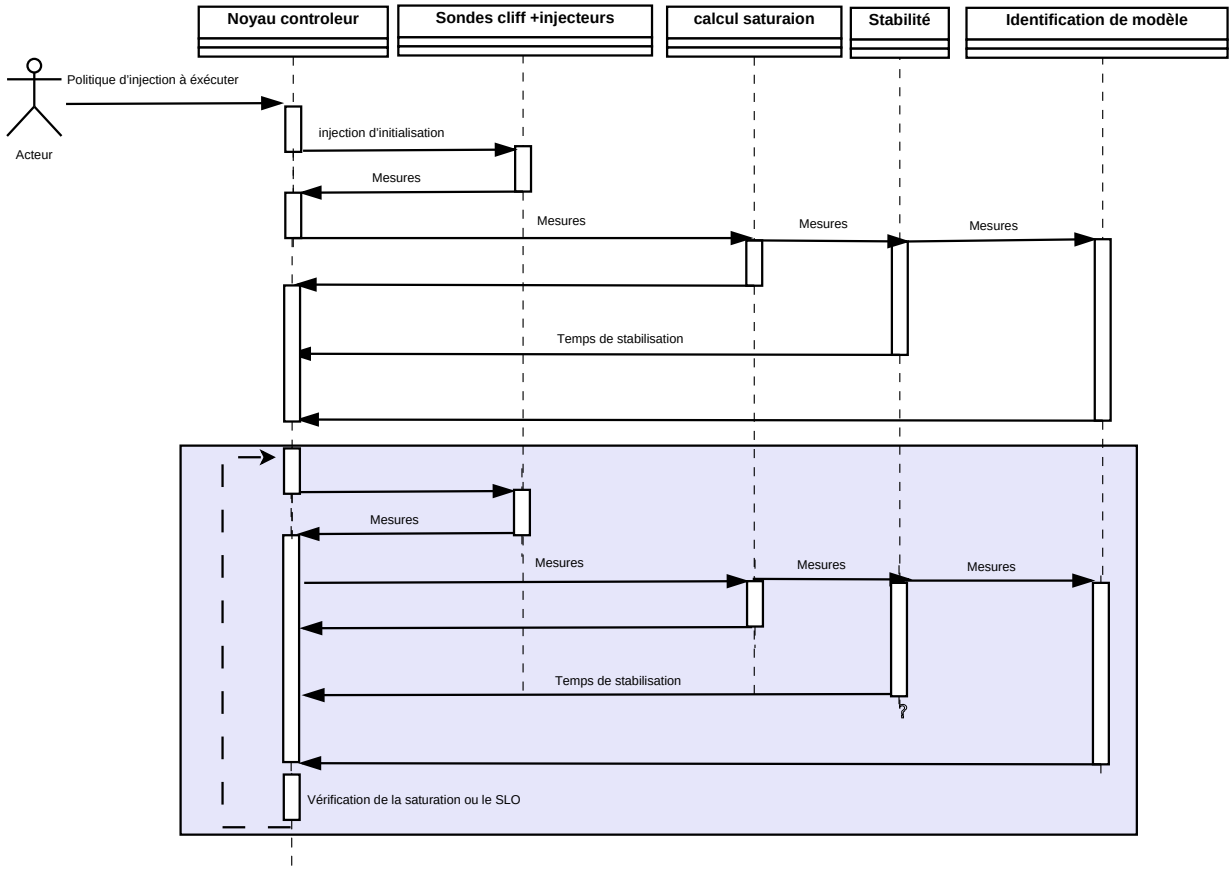

FIGURE 8.2 – Diagramme de séquence associé au scénario d'exécution

8.3.2 Architecture de l'environnement de résolution et de dimensionnement

a Vue fonctionnelle

La vue fonctionnelle associée à l'environnement de résolution et de dimensionnement est décrite dans la figure 8.3. On retrouve la base de modèles qui fournit les modèles des boîtes noires à combiner et à résoudre. Nous présentons dans la suite les autres modules de cet environnement, à savoir le module de configuration et le module de résolution.

Outil de configuration cet outil permet de choisir en premier une base de modèles à utiliser. Il choisit par la suite l'intervalle de niveau de charge à utiliser dans la phase de résolution. Ce choix permet de filtrer les modèles suivant la charge en entrée de chaque boîte afin de ne garder que les modèles validés pour ces niveaux de charge. Une fois sélectionnés, les modèles élémentaires constituent le système, une phase de paramétrage est lancée en définissant les interconnexions des boîtes, conformément à la configuration sous test : routage, nombre d'instances

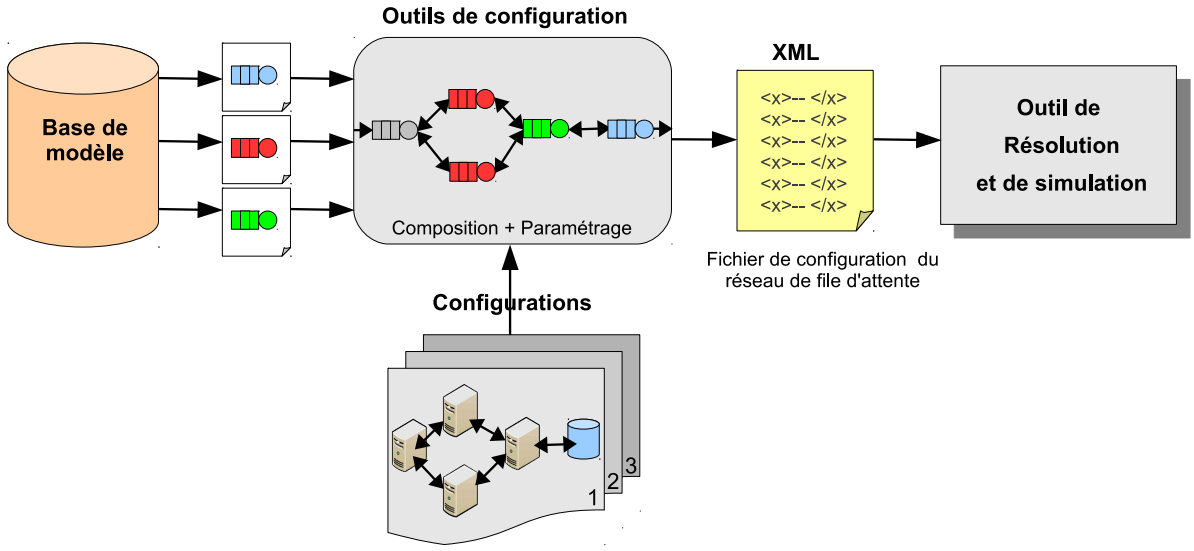

FIGURE 8.3 – Vue fonctionnelle de l'environnement de configuration et de dimensionnement

de chaque boîte et paramètres de chaque file d'attente. Le module permet aussi de définir le SLO à vérifier. Finalement, le fichier de configuration produit par ce module est une transcription de la configuration et de ces différents paramètres dans un format spécifique à l'outil de résolution.

Outil de résolution et de simulation cet outil récupère le fichier de configuration contenant la description du réseau de file d'attente à résoudre ou à simuler. Dans le cas de JMT, le fichier de configuration est défini en XML. Il contient à la fois des blocs de description graphique du réseau de file d'attente et des blocs de description des paramètres du modèle. Ce module permet également de choisir entre l'utilisation d'un algorithme de résolution ou la simulation, et de définir leurs paramètres, comme, par exemple, l'intervalle de confiance ou la durée de la simulation.

b Scénario d'exécution

L'expert commence par choisir une base de modèles associée à une campagne d'expérimentation. Une fois cette base chargée, il définit la valeur de la charge moyenne à l'entrée du système et commence par sélectionner le modèle de la première boîte. Il choisit alors le nombre d'instances, le taux d'inter-arrivées et le taux de service parmi les modèles validés. L'expert refait ces mêmes étapes avec toutes les boîtes formant le réseau. Une fois la configuration complètement définie, il choisit entre la résolution du réseau de files d'attente ou sa simulation. Le choix de la simulation lance une interface simplifiée du simulateur JSIM de JMT avec une représentation graphique du réseau de file d'attente utilisant JModel. L'expert a le choix de choisir entre fixer la durée de simulation ou utiliser la valeur par défaut avant de lancer de la simulation. La résolution par l'algorithme MVA lance une interface de configuration JMVA. Dans ce cas, l'expert peut modifier les classes de clients avant de lancer la résolution. Après résolution ou simulation, les résultats sont regroupés dans une interface à onglet : utilisation, débit, temps de

réponse et nombre de clients moyens dans chaque boîte et dans le réseau. En se basant sur ces résultats, l'expérimentateur peut vérifier le SLO et décider de l'adéquation de la configuration au système sous test pour la charge en entrée.

8.4 Implémentation

8.4.1 Expérimentation et modélisation automatique

a Implémentation en composants Fractal : FAMI

Fractal [22] est un modèle à composants modulaire et extensible, développé initialement par France Telecom et l'INRIA au sein du consortium OW2. Le modèle peut être utilisé indépendamment des langages de programmation pour concevoir, implémenter, déployer et reconfigurer différents systèmes et applications, allant des systèmes d'exploitation aux intergiciels et aux interfaces utilisateurs graphiques [2]. Il est utilisé dans divers projets, dont CLIF qui utilise Julia [21, 22] une des implémentations de Fractal en Java. Un composant Fractal est une entité de conception et d'exécution interagissant avec son environnement à travers des interfaces. Les interfaces peuvent être de deux types : une interface serveur, qui offre un service, et une interface cliente qui utilise un service. Un composant Fractal possède deux parties :

- la partie « contenu » constitue l'implémentation de la partie métier sous forme d'un ou plusieurs composants,
- la partie « contrôleur », implante des fonctions de contrôle telles que des fonctions d'interprétation des interactions avec ce composant ou de reconfiguration de propriétés internes au composant, ou la suspension et la reprise de son activité.

Fractal définit plusieurs interfaces de contrôle dans sa spécification. Un langage de description d'architecture Fractal ADL[20] basé sur le langage XML est utilisé pour décrire les configurations de composants Fractal tel que les interfaces, les liaisons, les attributs ou les définitions de contenus.

La figure 8.4 décrit l'architecture Fractal de CLIF avec des composants Fractal des blades (injecteurs et sondes) un composant superviseur, un composant de stockage et un composant d'analyse. Cette architecture étant modulaire et extensible nous cherchons à étendre l'architecture en ajoutant les composants nécessaires pour notre framework FAMI sans modifier les composants de CLIF. La figure 8.5 introduit une vue globale des composants ajoutés à l'architecture fractale de CLIF. Deux composants englobent tous les modules de contrôle et de modélisation de notre framework FAMI. Le premier composant est le composant «controller » qui utilise l'interface TestControl offerte par le superviseur. Cette interface sert notamment à consulter les mesures des lames, lire et changer leurs paramètres. D'autres méthodes seront ajoutées dans cette interface pour obéir aux besoins du contrôleur. La signature de cette interface «*TestControl.java*» est remplacée par «*BufferedTestControl.java*». Pour permettre à l'expérimentateur de

2. http ://Fractal.ow2.org

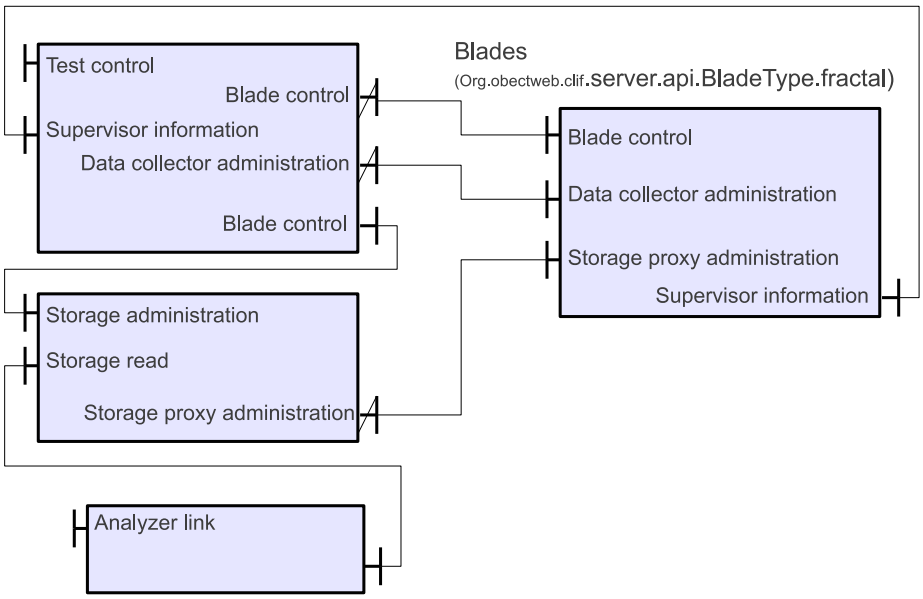

FIGURE 8.4 – Architecture Fractal de CLIF

suivre l'avancement de l'expérience de test en charge, on ajoute deux autres interfaces clientes au composant controller permettant de demander l'affichage dans la console de Supervision des décisions prises au cours de l'expérience (nombre d'utilisateurs qu'on décide d'ajouter, etc.) et des modèles de performance de la boîte qu'il calcule à chaque itération de contrôle. Cet affichage est dédié à un nouveau composant appelé «controller Monitor». Le composant «controller Monitor» effectue des affichages textuels sur une fenêtre de log de la console CLIF. Une interface graphique de monitoring utilisant plusieurs filtres est prévue pour mieux suivre l'expérimentation.

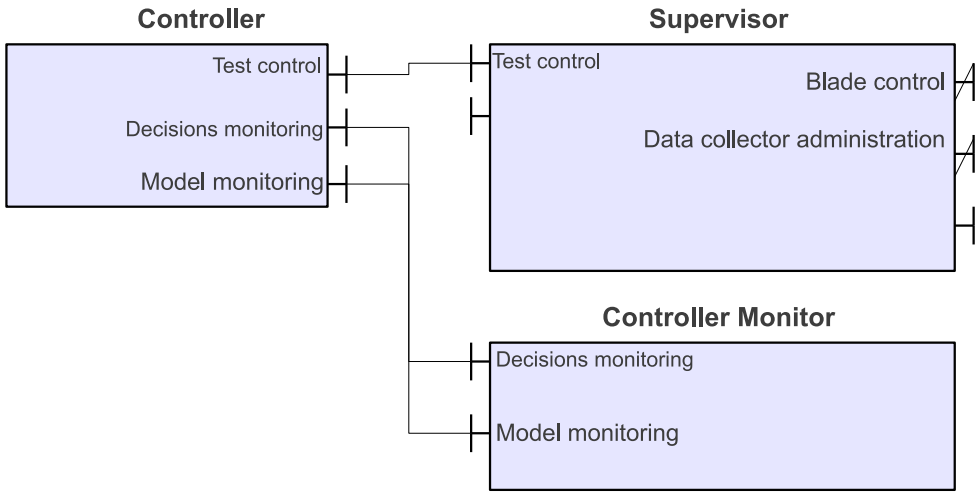

FIGURE 8.5 – Ajout des composants contrôleur et moniteur

Nous assemblons maintenant ces composants en modifiant le fichier Fractal ADL nommé «*ClifApp.fractal*» et en décrivant les liaisons (binding) entre ces composants conformément à la

figure 8.6.

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE definition SYSTEM "classpath://org/objectweb/fractal/adl/xml/basic.dtd">
<definition name="org.ow2.clif.console.lib.egui.ClifApp">

<component name="supervisor"      definition =
"org.ow2.clif.supervisor.api.BufferingSupervisorType" >
              <content class="org.ow2.clif.supervisor.lib.BufferingSupervisorImpl" />
 </component>
 <component name="storage"    definition="org.ow2.clif.storage.api.StorageType" >
               <content class="org.ow2.clif.storage.lib.filestorage.ConsoleFileStorageImpl" />
 </component>
<component name="controller"  definition ="org.ow2.clif.control.api.ControllerType" >
              <content class="org.ow2.clif.control.lib.ControllerImpl" />
 </component>
<component name="controller monitor" definition
="org.ow2.clif.control.monitor.api.contollerMonitor" />

<binding client="controller.Test control" server="supervisor.Test control" />
<binding client="controller.Decisions monitoring" server=" controller monitor.Decisions
monitoring " />
<binding client="controller.Model monitoring" server=" controller monitor.Model monitoring " />
 <binding client="supervisor.Storage administration" server="storage.Storage administration"/>
</definition>
```

FIGURE 8.6 – Modification du ClifApp.fractal et adaptation à la nouvelle architecture

Le contrôleur est défini dans le fichier ADL*ControllerType.fractal*, et implémenté par la classe *ControllerImpl.java*. La définition de ce composant et de ses interfaces est décrite dans la figure 8.7.

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE definition SYSTEM  "classpath://org/objectweb/fractal/adl/xml/core.dtd">

<definition name="org.ow2.clif.control.api.ControllerType">
<interface name = "Test control" role="client
          signature="org.ow2.clif.supervisor.api.BufferedTestControl" />
<interface name = " Decisions monitoring " role="client
          signature="org.ow2.clif.supervisor.api. DecisionsMonitoring " />
<interface name = " Model monitoring " role="client
          signature="org.ow2.clif.supervisor.api. ModelMonitoring " />
</definition>
```

FIGURE 8.7 – Définition et description des interfaces du composant "Controller"

Le composant «controller» est un composant composite qui englobe les calculs de temps de stabilisation, d'identification de modèles et l'identification du modèle global de la boîte à modéliser qui sont structurés dans des sous composants. L'implémentation actuelle regroupe le calcul de temps de stabilisation. Les tests d'hypothèseset l'estimation de la charge maximale se trouvent dans un même composant «computation». Ce composant est alors responsable des appels du framework R à travers son API R-Java. Cependant, ce même sous composant fera l'objet d'une séparation en sous composants conformément à la vue fonctionnelle décrite dans la

sous section a. Le sous composant «*model repository*» permet de stocker les modèles identifiés pour chaque palier. La figure 8.8 présente l'architecture Fractal du composant «controller».

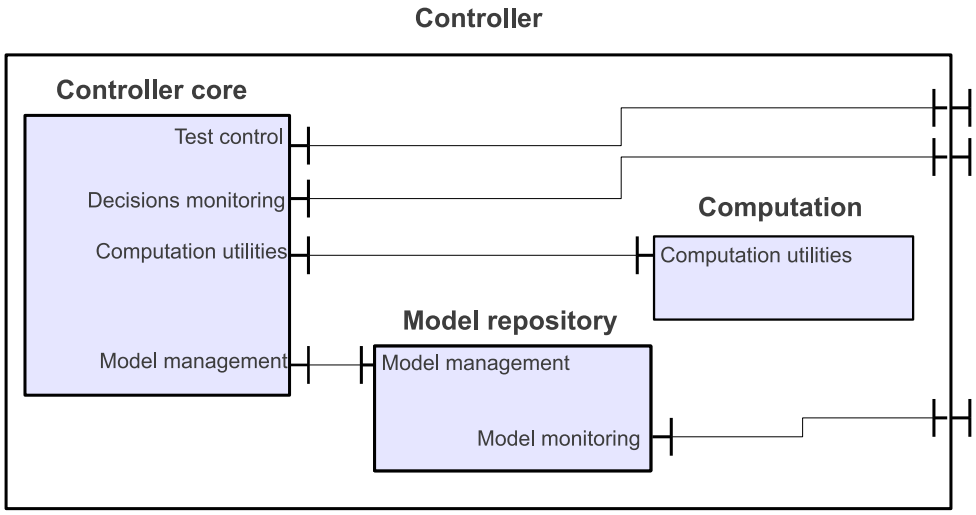

FIGURE 8.8 – Architecture Fractal du composant «Controller»

Le fichier qui représente l'architecture complète du contrôleur est alors un fichier ADL Controller.Fractal qui hérite de *ControllerType.fractal*. Dans ClifApp.fractal, il faut alors tenir compte de cette modification (changer ControllerType.fractal en Controller.fractal). Dans Fractal, la notion d'héritage permet à ajouter ou spécialiser les éléments existants de la définition présente (voir figure 8.9).

```
<definition name="org.ow2.clif.control.api.Controller"
extends="org.ow2.clif.control.api.ControllerType">

<component name="Controller core"  definition="org.ow2.clif.control.core.api.ControllerCore" />
<component name="Computation"
definition="org.ow2.clif.control.computation.api.Computation" />
<component name="Model repository"
definition="org.ow2.clif.control.Model.api.ModelRepository"/>

<!--  bindings between internal components -->
<binding client="Controller core.Model management"      server="Model repository.Model
management" />
<binding client="Controller core.Computation utilities" server="Computation.Computation
utilities"/>

<!--  bindings between main components & internal ones -->
<binding client="Controller core.Test control"        server="this.Test control"/>
<binding client="Controller core.Decisions monitoring" server="this.Decisions monitoring"/>
<binding client="Model repository.Model monitoring"    server="this.Model monitoring"/>

</definition>
```

FIGURE 8.9 – Fractal ADL du composant «Controller»

b Selfbench : deuxième implémentation

Selfbench[82] est une deuxième implémentation de notre approche de modélisation et de dimensionnement qui a été développée par l'équipe MAPS/MEP à Orange Labs. C'est une implémentation Java à base de composant Fractal qui a légèrement modifié la stratégie de montée en charge. En effet, l'injection de charge et la rétroaction ne se base plus sur un modèle de file d'attente identifié à la volée mais sur l'utilisation de statistiques glissantes sur les ressources (CPU, JVM et mémoire) et les temps de réponse. Cela réduit le temps d'attente associé à l'identification du modèle au niveau de chaque palier, réduisant ainsi le temps total de l'expérimentation. L'identification du modèle et le calcul de stabilité sont alors reportés après l'expérimentation et utilisent les mêmes algorithmes d'identification et de calcul. Cela modifie les paliers d'injection de charge. Cette modification réduit la qualité de montée en charge et augmente le risque de dépassement de la charge maximale. Cependant, pour des niveaux de charge rapprochés, les modèles de boîtes noires identifiés resteront les mêmes et pourront être utilisés pour la phase de dimensionnement. La motivation principale pour cette deuxième implémentation est dans les cas où l'utilisation réseau est critique (avec des débits d'injection de charge très importants). Dans ce cas, la rémontée intégrale des mesures brutes nécessaires au calcul du modèle et au temps de stabilisation peuvent saturer le réseau et altérer les résultats des expérimentations rendant le report des calculs après expérimentation une solution inévitable.

8.4.2 Configuration et dimensionnement

L'implémentation de l'outil de configuration utilise la technologie Java. Cet outil parcourt un répertoire dépôt contenant des fichiers XML qui récapitulent les résultats des expérimentations et les modèles identifiés. Ces modèles sont chargés dans l'interface utilisateur et classés suivant les boîtes noires et les intervalles de charge sur lesquel ces modèles sont valides. La figure 8.10 montre la définition d'une configuration de l'application *SampleCluster* utilisant deux serveurs Web, un serveur EJB et un serveur Mysql dans l'outil de configuration. La fonction « launch » permet de lancer soit la simulation soit l'analyse. Conformément à l'architecture décrite dans la sous section a, les outils de résolution et de simulation sont totalement indépendants et ils ne nécessitent aucune intégration. En effet, le modèle de la configuration à tester est produit dans un fichier de configuration de l'outil de résolution et de simulation. Actuellement, l'outil permet l'utilisation du solveur JMVA et du simulateur JSIM de JMT. Néanmoins, il est possible d'utiliser d'autres solveurs implémentant des algorithmes plus généraux, telque l'algorithme de Marie, ou d'implementer ces algorithmes dans JMT afin de couvrir d'autres types de systèmes.

Le choix d'une méthode de résolution produit un fichier *.jsimg*, format d'entrée pour JMT, et lance l'outil de résolution associée. La figure 8.11 montre la représentation de la configuration de *SampleCluster* sur l'interface de modélisation JModel. L'expert peut définir un temps d'exécution maximal avant de lancer la simulation.

La figure 8.12 montre les résultats finaux de la simulation. Ces résultats sont calculés sur la base d'un échantillon de taille 66560 (case réduite sur la figure). Plusieurs paramètres globaux

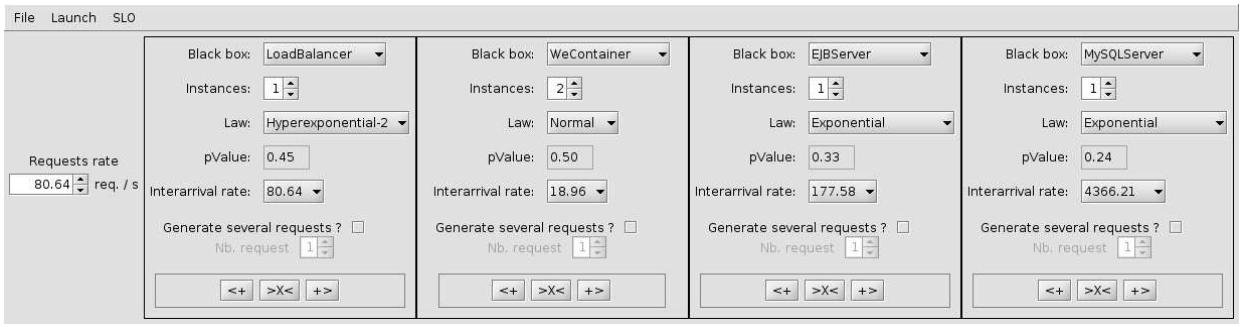

FIGURE 8.10 – Outil de configuration : choix de la configuration à étudier

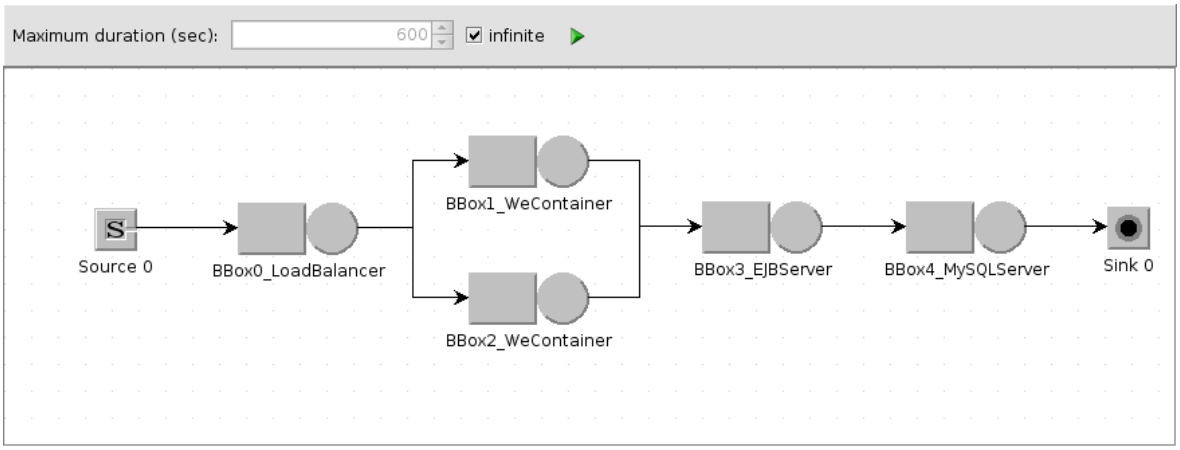

FIGURE 8.11 – JModel : réseau de file d'attente associé à la reconfiguration à simuler

ou par station sont calculés avec un intervalle de confiance à 95%. Les paramètres qui nous intéressent dans cet exemple sont le temps de réponse du réseau de file d'attente *system response time* et le taux d'utilisation. Le temps de réponse sera comparé avec le SLO défini pour décider de l'acceptation de la reconfiguration. Si cette reconfiguration est retenue, elle doit être comparée avec les autres reconfigurations afin de choisir une solution optimale à moindre coût. Le taux d'utilisation associé à chaque station de file d'attente permet d'éliminer les reconfigurations ayant une station avec une forte utilisation. Cela permet d'éviter de créer des éventuels goulets d'étranglement.

Dans le cas où les modèles de file d'attente permettent de faire une analyse MVA (voir figure 8.13), les paramètres d'utilisation et de temps de réponse par station sont calculés. Il suffit alors d'utiliser ces valeurs pour déterminer une estimation du temps de réponse du réseau de file d'attente et de procéder de la même manière qu'avec les résultats de la simulation.

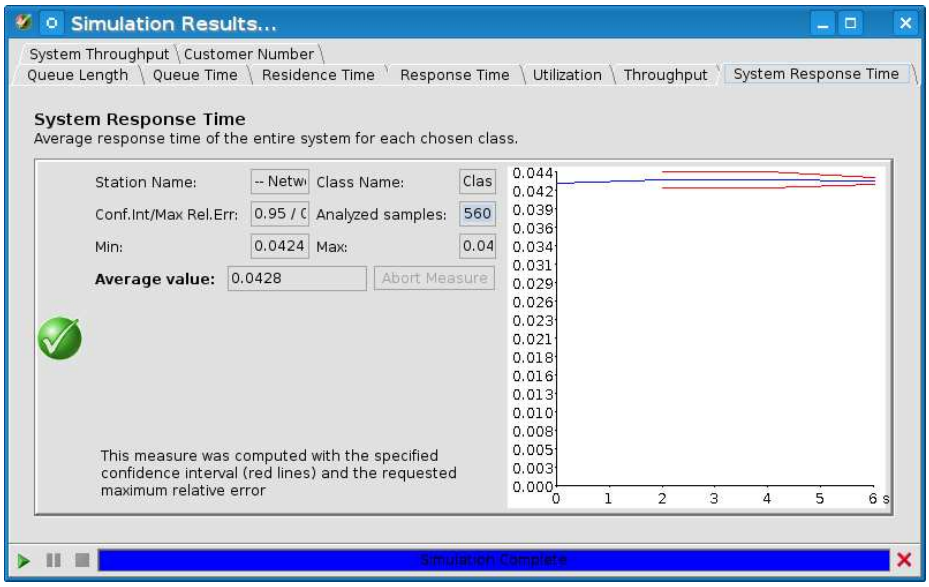

FIGURE 8.12 – JSIM : estimation du temps de service du réseau de file d'attente par simulation

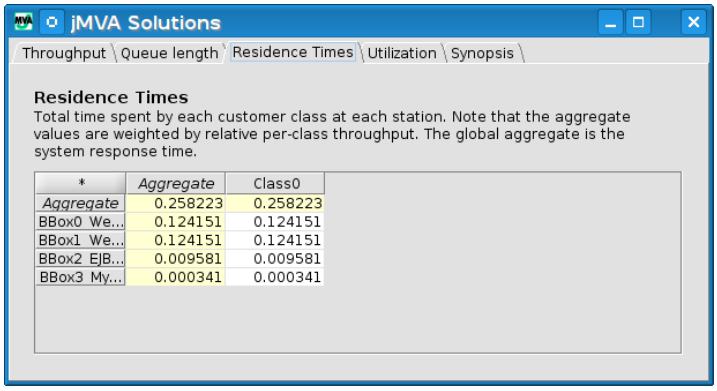

FIGURE 8.13 – JMVA : calcul de temps de séjour par station par analyse MVA

8.5 Evaluation de l'architecture

Afin d'évaluer l'architecture de notre framework, nous allons vérifier l'ensemble des critères et des besoins définis au début de ce chapitre. Du point de vue modularité, notre architecture utilise le modèle à composant Fractal. L'utilisation de ce modèle permet de structurer notre architecture dans des composants regroupant un ensemble de traitements suivant une vue logique. Cela permet, en cas de besoin, une restructuration simple de l'application. Cette restructuration se fait par simple modification d'un fichier de configuration Fractal ADL.

L'utilisation de CLIF et de ses plugins donne plus de généricité et d'extensibilité à ce framework. Tout système distribué peut alors être injecté et instrumenté. En cas d'absence de plugins dédiés au protocole de communication, l'utilisateur peut développer et utiliser son propre plugin d'injection. La politique d'injection est également générique grâce au langage de description de

la politique d'injection au format xml. Ce langage permet d'exprimer les politiques d'injection les plus complexes, en utilisant les structures de contrôle et en utilisant des variables de contrôle de l'injection et même des appels de fonction statistique à partir de R. L'extensibilité de ce framework est aussi assurée par l'utilisation des projets logiciels libres comme R et JMT, ce qui rend la tâche d'ajouter des algorithmes de résolution des réseaux de files d'attente ou des tests d'hypothèse possible. Le Framework est éfficace dans le sens où il permet de réduire considérablement le temps d'expérimentation et permet de déterminer les modèles des boîtes en temps acceptable.

Cependant, le framework présente aussi des limites. L'outil de configuration ne permet que la description des architectures à étage de type multi-tiers. Cela réduit les types de système pouvant être évalué. Cet outil pourraît être revu afin de permettre de concevoir différents types de configuration à partir des modèles des boîtes identifiées. L'utilisation de l'outil de représentation des modèles JModel de JMT est envisageable. D'autres algorithmes de résolution plus généraux doivent être implémentés afin de pouvoir résoudre tout type de système sans être obligé de passer par la simulation. L'implémentation de ces algorithmes dans JMT donnera la possibilité d'intégrer complètement ce framework afin de produire une seule unité logicielle de configuration et de dimensionnement. Finalement, le calcul actuel de temps de stabilisation pour des modèles complexes peut engendrer des pertes considérables de temps de calcul. Pour cela il faut prévoir d'ajouter un seuil sur les durées de calcul des temps de stabilisation. En cas de dépassement, une borne maximale de temps de stabilisation sera appliquée pour éviter tout blocage éventuel.

8.6 Conclusion

Dans ce chapitre, nous avons montré les aspects fonctionnels et architecturaux de notre environnement d'expérimentation et de modélisation automatique. Nous avons aussi utilisé et intégré d'autres logiciels libre (JMT, R) permettant de compléter la phase de résolution et de dimensionnement en prenant en considération les différents besoins techniques et fonctionnels décrit au début de ce chapitre.

Finalement, nous avons évalué l'architecture et l'implémentation de notre framework en déterminant ses limites. L'ensemble de ces limites, ainsi que, ainsi que d'autres, seront reprises dans le chapitre suivant, en présentant des solutions envisageables et des perspectives pour ce travail.

Conclusion et perspectives

La problématique abordée dans cette thèse est le dimensionnement des systèmes répartis. On propose une méthodologie complète de dimensionnement qui repose principalement sur l'automatisation des étapes d'expérimentation et de modélisation des composants « boîtes noires» formant le système. Un *framework* a été développé à cet effet permettant, d'une part, d'automatiser la modélisation des boîtes noires, et d'autres part, de fournir des outils d'aide au dimensionnement.

Notre approche repose sur l'utilisation de boucles de contrôle permettant l'identification progressive d'un modèle de performances et de contrôler les paramètres de l'expérimentation. D'un côté, chaque cycle de la boucle de contrôle permet d'avoir plus de mesures fiables pour une meilleure identification de modèle. Et de l'autre côté, le modèle est utilisé pour paramétrer les prochaines étapes de l'expérimentation. Ce contrôle permet de calculer le temps de stabilisation et de déterminer les limites opérationnelles de chaque composant du système. Notre approche repose sur une identification du modèle basée uniquement sur les mesures et les tests d'hypothèse statistique. Cela implique qu'on ne se base ni sur des connaissances des mécanismes internes du système ni sur des fortes hypothèses sur les distributions utilisées. Les modèles identifiés sont des modèles ayant un spectre large de distributions de service et des inter-arrivées qui peuvent dans plusieurs cas enrichir nos connaissances sur la nature du service ou sur ses performances. En effet, notre processus d'identification a mis en évidence, à partir d'un niveau de charge, des modèles avec des distributions de services Hyper-exponentielles. Ces distributions peuvent être interprétées soit par l'existence de deux types de service avec des temps de réponse moyens différents, soit par des rejets des requêtes générant ainsi un flux de clients avec un temps de réponse moyen inférieur au temps de service normal.

Les étapes d'expérimentation et de modélisation automatique sont appliquées à tous les composants formant le système. Ces composants sont obtenus après une décomposition adaptée au contexte distribué des applications et aux problèmes liés au dimensionnement. L'ensemble des modèles de ces composants est rassemblé dans une boîte à outils permettant à l'expérimentateur de :

- composer le modèle global du système existant et évaluer ses performances en vérifiant le *Service Level Objectives*.
- composer des modèles d'optimisation ou de réparation afin de reconfigurer le système.
- analyser ou de simuler plusieurs configurations possibles pour choisir une solution qui

répond au mieux aux problèmes de performances, au moindre coût.

L'approche a été implémentée en utilisant le logiciel libre d'injection de charges répartie CLIF. Les étapes d'expérimentation et de modélisation ont été entièrement automatisées et ont permis de réduire considérablement le temps d'expérimentation sans réduire la qualité des mesures ou des modèles générés. Ce *framework* fournit aussi à l'administrateur des outils d'aide au dimensionnement. En effet, il a été couplé à un générateur de configuration et un environnement d'analyse et de simulation des réseaux des files d'attente basé sur le logiciel libre JMT. L'expérimentateur dispose ainsi d'une solution complète d'évaluation de performances et de dimensionnement.

Des améliorations des ces travaux porteront ultérieurement sur :

Du point de vue approche :

1. L'approche se base sur des reconfigurations proposées par l'expérimentateur. Pour automatiser cette phase et permettre le dimensionnement dynamique, on doit développer un générateur automatique des configurations et un outil de décision. Ces outils se basent sur le calcul de la demande et des capacités de traitement de chaque ressource ou de chaque composant. Plusieurs configurations candidates doivent être générées en proposant des réplications d'éléments du système. L'analyse ou la simulation de ces configurations permet de déterminer la meilleure solution à déployer.
2. Le calcul de temps de stabilisation est actuellement approché par le temps de stabilisation d'une file M/M/1/C puis corrigé expérimentalement. La vitesse de convergence de cette file vers son régime stationnaire dépend de l'incrément choisi (le pas d'injection) et de la charge $\boldsymbol{c}$. Lorsque ces deux valeurs sont grandes, le temps de calcul du temps de stabilisation croît considérablement, ce qui représente un blocage potentiel du processus automatique de l'expérimentation. On propose de définir un seuil du temps de calcul à partir duquel on utilise une valeur approchée du temps de stabilisation. Cette valeur approchée est déterminée à partir d'un tableau associant à chaque modèle et à chaque charge $\boldsymbol{c}$ un temps de stabilisation calculé à l'avance.

Du point de vue *framework* :

1. Contrairement à notre approche valable pour un grand nombre de systèmes répartis, l'outil de configuration actuel ne permet que la formalisation des architectures multi-tiers. L'outil *JMT* fournit une interface *JModel* capable de modéliser des architectures de réseaux de files d'attente diverses et variées. On propose d'utiliser *JModel* en ajoutant une boîte à outil dans l'interface graphique. Cette boîte à outils fournit à l'expérimentateur l'ensemble des modèles de boîtes noires identifiées afin de former différentes architectures possibles.
2. L'implémentation d'autres algorithmes de résolution, conformément aux diagrammes 2.5 et 2.9, permet d'élargir les types de réseaux de files d'attente pris en compte par l'analyse. Ces algorithmes, qui à implémenter dans *JMT*, permettent de réduire le temps de résolution. De plus, l'expérimentateur aura le choix entre des solutions approximatives avec un temps de résolution rapide, ou inversement selon les besoins de dimensionnement.

Plusieurs perspectives peuvent être envisagées :

1. L'adaptation et l'intégration de notre *framework* dans une plate-forme de type *Cloud computing*. Cette perspective s'inscrit dans le contexte du dimensionnement automatique et de déploiement des solutions logicielles en mode PaaS (Platform as a service). En offrant des environnements d'exécution pour ses clients, les fournisseurs du service s'engagent à assurer une qualité de service permanente. D'où la nécessité de prédire les performances demandés des services clients. L'utilisation de notre approche de dimensionnement permettra aux fournisseurs du *Cloud* de se baser non seulement sur l'utilisation des ressources mais aussi sur un modèle spécifique à l'application cliente et son environnement. Nous proposons aussi d'intégrer ces travaux à l'environnement open-source *OpenNEbula*[89] et au front-end de provisionning *Haizea*[88]. Ceci permettrait de fournir un outil de décision qui présenterait une continuité aux travaux de provisionning dans les plateformes Paas [87] ;
2. Etude de stabilisation du système à forte charge : l'approximation utilisée dans cette thèse doit être affinée. En effet, la correction expérimentale du temps de stabilisation devient inutile dans les conditions des fortes charges du fait de la forte variation des temps de réponse. Dans ces conditions seule l'approximation théorique compte. L'étude de temps de stabilisation dans le cas de fortes charges reste un problème ouvert et à traiter spécifiquement au contexte des applications réparties actuelles.
3. Le trafic étudié dans cette thèse est un trafic homogène adapté à un contexte spécifique. Une extension de ces travaux est d'adapter notre approche au contexte multi-classes des clients. Ce changement de contexte induit un changement de méthodologie. Les modifications commencent par la classification du trafic suivant les différents profiles de consommation de ressources (processeur, disque, etc). Les modifications concernent également la nature de la charge expérimentale à injecter et les algorithmes de résolutions utilisés.

Bibliographie

[1] Gestion du trafic : Modèles pour les réseaux de télécommunications.

[2] Le petit lexique des termes de la complexité.

[3] *Brouwer theorem, Encyclopaedia of Mathematics*. Kluwer Academic Publishers, 2002.

[4] T.F. Abdelzaher, K.G. Shin, and N. Bhatti. Performance guarantees for web server end-systems : A control-theoretical approach. *IEEE Trans. Parallel Distrib. Syst.*, 13(1) :80–96, 2002.

[5] C. Amza, A. Chanda, A.L. Cox, S. Elnikety, R. Gil, K. Rajamani, W. Zwaenepoel, E. Cecchet, and J. Marguerite. Specification and implementation of dynamic web site benchmarks. In *Workload Characterization, 2002. WWC-5. 2002 IEEE International Workshop on*, pages 3–13. IEEE, Nov. 2002.

[6] J. Arnaud. Automated control of internet services. In *Proceedings of the Fifth International Workshop on Feedback Control Implementation and Design in Computing Systems and Networks*, pages 7–12. ACM, 2010.

[7] J. Arnaud and S. Bouchenak. Adaptive internet services through performance and availability control. In *Proceedings of the 2010 ACM Symposium on Applied Computing*, pages 444–451. ACM, 2010.

[8] G. Balbo and G. Serazzi. Asymptotic analysis of multiclass closed queueing networks : Common bottleneck* 1. *Performance Evaluation*, 26(1) :51–72, 1996.

[9] G. Balbo and G. Serazzi. Asymptotic analysis of multiclass closed queueing networks : Multiple bottlenecks* 1. *Performance Evaluation*, 30(3) :115–152, 1997.

[10] Y. Bard. Some extensions to multiclass queueing network analysis. In *Proceedings of the Third International Symposium on Modelling and Performance Evaluation of Computer Systems : Performance of Computer Systems table of contents*, pages 51–62. North-Holland Publishing Co. Amsterdam, The Netherlands, The Netherlands, 1979.

[11] F. Baskett, K.M. Chandy, R.R. Muntz, and F.G. Palacios. Open, closed, and mixed networks of queues with different classes of customers. *Journal of the ACM (JACM)*, 22(2) :248–260, 1975.

[12] B. Baynat. *Théorie des files d'attente, des chaînes de Markov aux réseaux à forme produit*. Hermes Science Publications, Paris, 2000.

[13] M. Bertoli, G. Casale, and G. Serazzi. Java modelling tools, 2006. http ://jmt.sourceforge.net/.

[14] M. Bertoli, G. Casale, and G. Serazzi. Java modelling tools : an open source suite for queueing network modelling and workload analysis. In *Proceedings of QEST 2006 Conference*, pages 119–120, Riverside, US, Sep 2006. IEEE Press. http ://jmt.sourceforge.net.

[15] M. Bertoli, G. Casale, and G. Serazzi. The jmt simulator for performance evaluation of non-product-form queueing networks. In *Annual Simulation Symposium*, pages 3–10, Norfolk,VA, US, 2007. IEEE Computer Society. http ://doi.ieeecomputersociety.org/10.1109/ANSS.2007.41.

[16] M. Bertoli, G. Casale, and G. Serazzi. Jmt : performance engineering tools for system modeling. *SIGMETRICS Perform. Eval. Rev.*, 36(4) :10–15, 2009. http ://doi.acm.org/10.1145/1530873.1530877.

[17] G. Bolch. Leistungsbewertung von Rechensystemen. *Teubner-Verlag, Stuttgart*, 1989.

[18] G. Bolch, S. Greiner, H. de Meer, and K. S. Trivedi. *Queueing networks and Markov chains. modelling and Performance Evaluation with Computer Science Applications*. JOHN WILEY and SONS, Canada, 2006.

[19] G. Bolch, S. Greiner, H. de Meer, and K.S. Trivedi. *Queueing networks and Markov chains*. Wiley-Interscience, 2005.

[20] E. Bruneton. Fractal adl tutorial 1.2. *France Telecom R&D*, 2004.

[21] E. Bruneton, T. Coupaye, M. Leclercq, V. Quéma, and J.B. Stefani. An open component model and its support in java. *Component-Based Software Engineering*, pages 7–22, 2004.

[22] E. et al. Bruneton. The fractal component model and its support in java. *Software Practice and Experience, special issue on Experiences with Auto-adaptive and Reconfigurable Systems*, 36(11-12), 2006.

[23] R.M. Bryant, A.E. Krzesinski, M.S. Lakshmi, and K.M. Chandy. The MVA priority approximation. *ACM Transactions on Computer Systems (TOCS)*, 2(4) :359, 1984.

[24] K.M. Chandy and D. Neuse. Linearizer : a heuristic algorithm for queueing network models of computing systems. *ACM*, 1982.

[25] W. Chiu. Design for scalability-an update, 2001. http ://www.ibm.com/developerworks/websphere/library/techarticles/hipods/scalability.html.

[26] P. Chylla. Zur Modellierung und approximativen Leistungsanalyse von Vielteilnehmer-Rechensystemen. *Dessertation, Faculty for Mathematics and Computer Science, Technical university Munich*, 1986.

[27] OW2 Consortium. Jasmine, the smart tool for your soa platform management, 2011. http ://jasmine.ow2.org.

[28] J.M. Cornu. *ProspecTIC. Nouvelles technologies, nouvelles pensées*. FVP Editions, 2008.

[29] R.B. D'Agostino and M.A. Stephens. *Goodness-of-fit techniques*. CRC, 1986.

[30] B. Dillenseger. Clif, a framework based on fractal for flexible, distributed load testing. *Annals of Telecommunications*, 64 :101–120, 2009. 10.1007/s12243-008-0067-9.

[31] B. Dillenseger. Clif, a framework based on fractal for flexible, distributed load testing. *Annals of Telecom*, 64(1-2) :101–120, Feb. 2009.

[32] Varoquaux E. Ecriture en java d'un moteur d'exécution optimisé de scénarios d'injection de charge pour une plate-forme de mesure de performances. Rapport de stage de fin d'études Ensimag, 2005.

[33] D.L. Eager, D.J. Sorin, and M.K. Vernon. AMVA techniques for high service time variability. In *Proceedings of the 2000 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, pages 217–228. ACM New York, NY, USA, 2000.

[34] E. Gelenbe, G. Pujolle, and JCC Nelson. *Introduction to queueing networks*. John Wiley & Sons, Inc. New York, NY, USA, 1987.

[35] A. Gilat. *MATLAB : An Introduction with Applications 2nd Edition*. Wiley, 2004.

[36] P. Ginger, B.and Raymond and Oh N.H. Loadrunner. software and website, 2007. http ://environment.yale.edu/raymond/loadrunner/.

[37] W. Gordon and G. Newell. Closed queuing systems with exponential servers. *Operations Research*, pages 254–265, April 1967.

[38] D. Gross. *Fundamentals of Queueing Theory*. Wiley-Interscience, New York, 2008.

[39] E.H. Halili. *Apache JMeter : A practical beginner's guide to automated testing and performance measurement for your websites*. Packt Publishing, 2008.

[40] IBM. An architectural blueprint for autonomic computing. http ://www-03.ibm.com/autonomic/pdfs/AC Blueprint White Paper V7.pdf, June 2005.

[41] R. Ihaka and R. Gentleman. R : A language for data analysis and graphics. *Journal of Computational and Graphical Statistics*, 5(3) :299–314, 1996.

[42] NETWORK INSTRUMENTS. Retrospective network analysis. Technical report, NETWORK INSTRUMENTS, 2007.

[43] J.R. Jackson. Networks of waiting lines. *Operations Research*, 5(4) :518–521, 1957.

[44] R. K. Jain. *The Art of Computer Systems Performance Analysis : Techniques for Experimental Design, Measurement, Simulation, and modelling*. John Wiley and Sons, Inc, Canada, April 1991.

[45] G. Jung, G. Swint, J. Parekh, C. Pu, and A. Sahai. Detecting bottleneck in n-tier it applications through analysis. *Large Scale Management of Distributed Systems*, pages 149–160, 2006.

[46] A. Kamra, V. Misra, and E.M. Nahum. Yaksha : a self-tuning controller for managing the performance of 3-tiered web sites. In *IWQoS*, pages 47–56, 2004.

[47] M. Karlsson and M. Covell. Dynamic black-box performance model estimation for self-tuning regulators. In *ICAC '05 : Proceedings of the Second International Conference on Automatic Computing*, pages 172–182, Washington, DC, USA, 2005. IEEE Computer Society.

[48] J. O. Kephart and D. M. Chess. The vision of autonomic computing. *Computer*, 36(1) :41–50, January 2003.

[49] L. Kleinrock. *Queueing systems. Vol. II : Computer applications*. New York etc, 1976.

[50] Leonard Kleinrock. *Queueing Systems*. Wiley-Interscience, New York, 1975.

[51] P. Kraus. *Prévision et maîtrise des performances d'un système informatique*. Hermes-Lavoisier, 2005.

[52] P. Kuehn. Approximate analysis of general queuing networks by decomposition. *Communications, IEEE Transactions on*, 27(1) :113–126, Jan 1979.

[53] P. Kuehn and G. Siegen. Approximate analysis of general queuing networks by decomposition. *Communications, IEEE Transactions on [legacy, pre-1988]*, 27(1) :113–126, 1979.

[54] P. Lafaye de Micheaux, R. Drouilhet, and B. Liquet. *Le Logiciel R. Maîtriser le langage, effectuer des analyses statistiques*. Springer, Collection Statistiques et Probabilités appliquées, 1st edition, 2010. ISBN : 9782817801148.

[55] A. Law and W.D. Kelton. *Simulation Modeling and Analysis (Industrial Engineering and Management Science Series)*. McGraw-Hill Science/Engineering/Math, 1999.

[56] J.Y Le Boudec. *Performance Evaluation Lecture Notes (Methods, Practice and Theory for the Performance Evaluation of Computer and Communication Systems*. EPFL, 2009.

[57] J.P. Lecoutre and P. Tassi. *Statistique non paramétrique et robustesse*. Économica, 1987.

[58] L.R. Lipsky. *Queuing theory : A linear algebraic approach*. Springer Verlag, 2008.

[59] M Litoiu. A performance analysis method for autonomic computing systems. *TAAS*, 2(1), 2007.

[60] J.D. C. Little. A proof for the queuing formula : L= λw. *Operations Research*, pages 383–387, May-June 1961.

[61] Y. Lu, T. Abdelzaher, Ch. Lu, L Sha, and X. Liu. Feedback control with queueing-theoretic prediction for relative delay guarantees in web servers. In *RTAS '03 : Proceedings of the The 9th IEEE Real-Time and Embedded Technology and Applications Symposium*, page 208, Washington, DC, USA, 2003. IEEE Computer Society.

[62] S. Malkowski, M. Hedwig, J. Parekh, C. Pu, and A. Sahai. Bottleneck detection using statistical intervention analysis. *Managing Virtualization of Networks and Services*, pages 122–134, 2007.

[63] R. Marie. Calculating Equilibrium Probabilities For X(n)/Ck/l/N Queues. In *ACM Sigmetrics Performance Evaluation Review*, volume 9, pages 117–125. ACM, 1980.

[64] RA Marie. An approximate analytical method for general queueing networks. *IEEE Transactions on Software Engineering*, pages 530–538, 1979.

[65] D. A. Menascé, D. Barbará, and R. Dodge. Preserving qos of e-commerce sites through self-tuning : a performance model approach. In *EC '01 : Proceedings of the 3rd ACM conference on Electronic Commerce*, pages 224–234, New York, NY, USA, 2001. ACM Press.

[66] D.A. Menasce and V. A.F. Almeida. *Capacity Planning for Web Services : metrics, models, and methods*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2001.

[67] D.A. Menasce, V. A.F. Almeida, and L. W. Dowdy. *Capacity Planning and Performance Modeling : From Mainframes to Client-Server Systems*. Prentice Hall, 1994.

[68] D.A. Menasce, M.N. Bennani, and H. Ruan. On the use of online analytic performance models in self-managing and self-organizing computer systems.

[69] G. Millot. *Comprendre et réaliser les tests statistiques avec R : Manuel pour les débutants*. De Boeck Université, 2009.

[70] S. Morgenthaler. *Introduction à la statistique*. PPUR presses polytechniques, 2007.

[71] K. Nagaraja, F Oliveira, R Bianchini, R.P. Martin, and Thu D. Nguyen. Understanding and dealing with operator mistakes in internet services. In *OSDI04 : In Proceedings of the 6th conference on Symposium on Opearting Systems Design and Implementation*, pages 1–76, Berkeley, CA, USA, 2004. USENIX Association.

[72] D. Neuse and K. Chandy. SCAT : A heuristic algorithm for queueing network models of computing systems. *ACM Sigmetrics Performance Evaluation Review*, 10(3) :59–79, 1981.

[73] UK Newcastle University. Javasim, 1997. http ://javasim.codehaus.org/.

[74] J. Parekh, G. Jung, G. Swint, C. Pu, and A. Sahai. Issues in bottleneck detection in multi-tier enterprise applications. In *Quality of Service, 2006. IWQoS 2006. 14th IEEE International Workshop on*, pages 302–303. IEEE, 2006.

[75] G. Pujolle and W. Ai. A solution for multiserver and multiclass open queueing networks. *Information Systems and Operations Research*, 24(3) :221–230, 1986.

[76] F. Rafamantanantsoa, P. Laurenot, and A. Aussem. Analyse, modélisation et contrôle en. Technical Report LIMOS/RR-05-06, LIMOS, 10 Mars 2005.

[77] D. S. Reginald. The dynamics of internet traffic : Self-similarity, self-organization, and complex phenomena. *CoRR*, abs/0807.3374, 2008.

[78] M. Reiser. Mean value analysis of queueing networks, a new look at an old problem. *4th Int. Symp. on Modelling and performance Evaluation of computer Systems*, pages 63–67, 1979.

[79] M. Reiser and H. Kobayashi. On the convolution algorithm for separable queuing networks. In *Proceedings of the 1976 ACM SIGMETRICS conference on Computer performance modeling measurement and evaluation*, pages 109–117. ACM New York, NY, USA, 1976.

[80] M. Reiser and S.S. La Venberg. Mean value analysis of closed multichain queueing networks. *J. ACM*, pages 313–323, 1980.

[81] S.I. Resnick. Heavy tail modeling and teletraffic data. *Annals of Statistics*, 25 :1805–1869, 1997.

[82] N Salmi, A Harbaoui, B Dillenseger, and J.M Vincent. Model-Based Performance Anticipation in Multi-tier Autonomic Systems : Methodology and Experiments. *International Journal On Advances in Networks and Services*, 3(3-4), 2010.

[83] G. Saporta. *Probabilités, analyse des données et statistique*. Editions Technip, 2006.

[84] P. Schweitzer. Approximate analysis of multiclass closed networks of queues. In *International conference on stochastic control and optimization*, pages 25–29, 1979.

[85] K. Sevcik and I. Mitarni. The distribution of queuing network states at input and output instants. *J. ACM vol. 28 no. 2*, April 1981.

[86] SIMULOG. QNAP2 tool, 1985. http ://www.simulog.fr.

[87] B. Sotomayor, R.S. Montero, I.M. Llorente, I. Foster, and F. de Informatica. Capacity leasing in cloud systems using the opennebula engine. *Cloud Computing and Applications*, 2008 :1–5, 2008.

[88] Borja Sotomayor, Kate Keahey, and Ian Foster. Combining batch execution and leasing using virtual machines. In *Proceedings of the 17th international symposium on High performance distributed computing*, HPDC '08, pages 87–96, New York, NY, USA, 2008. ACM.

[89] Borja Sotomayor, Ruben S. Montero, Ignacio M. Llorente, and Ian Foster. Virtual infrastructure management in private and hybrid clouds. *IEEE Internet Computing*, 13 :14–22, 2009.

[90] P. Sprent and J.P. Ley. *Pratique des statistiques nonparamétriques*. Editions Quae, 1992.

[91] W. Stewart. *Introduction to the Numerical Solution of Markov Chains*. Princeton University Press, Princeton, 1994.

[92] Morris W. Sturm, R. and M. Jander. *Foundations of Service Level Management*. Sams Publishing, Indianapolis, 2003.

[93] G.S. Swint, G. Jung, C. Pu, and A. Sahai. Automated staging for built-to-order application systems. In *Network Operations and Management Symposium, 2006. NOMS 2006. 10th IEEE/IFIP*, pages 361–372. IEEE, 2006.

[94] M. Tenenhaus. *Statistique : Méthodes pour décrire, expliquer et prévoir*. Dunod, 2007.

[95] B.W. Turnbull. The empirical distribution function with arbitrarily grouped, censored and truncated data. *Journal of the Royal Statistical Society. Series B (Methodological)*, 38(3) :290–295, 1976.

[96] B. Urgaonkar, P. J. Shenoy, A. Chandra, and Goyal P. Dynamic provisioning of multi-tier internet applications. In *ICAC*, pages 217–228, 2005.

[97] G. Urroz. *Numerical and Statistical Methods with SCILAB for Science and Engineering - Volume 2*. BookSurge Publishing, 2001.

[98] L. Verlaine, F. Hardange, F. Biard, and D. Elias. *Tests de performance des applications Web : retours d'expérience : l'usine e-testing de France Télécom (Solutions d'entreprise)*. Eyrolles, 2003.

[99] H. Wang. *Approximate MVA algorithms for solving queueing network models*. PhD thesis, Citeseer, 1997.

[100] W. Whitt. The queueing network analyzer. *Bell System Technical Journal*, 62(9) :2817–2843, 1983.

[101] W. Whitt. The queueing network analyzer. *Bell System Technical Journal*, 62(9) :2779–2815, 1983.

[102] M. Woodside, T. Zheng, and M. Litoiu. Performance model estimation and tracking using optimal filters. *IEEE Transactions on Software Engineering*, 34(3) :391–406, 2008.

[103] L. Yin, S. Uttamchandani, and R. Katz. An empirical exploration of black-box performance models for storage systems. In *MASCOTS '06 : Proceedings of the 14th IEEE International Symposium on modelling, Analysis, and Simulation*, pages 433–440, Washington, DC, USA, 2006. IEEE Computer Society.

[104] J. Zahorjan, K.C. Sevcik, D.L. Eager, and B. Galler. Balanced job bound analysis of queueing networks. *ACM*, 1982.

[105] B.P. Zeigler, H Praehofer, and T.G Kim. *Theory of Modeling and Simulation, Second Edition*. Academic Press, 2000.

[106] T. Zheng, J. Yang, M. Woodside, M. Litoiu, and G. Iszlai. Tracking time-varying parameters in software systems with extended kalman filters. In *CASCON '05 : Proceedings of the 2005 conference of the Centre for Advanced Studies on Collaborative research*, pages 334–345. IBM Press, 2005.

Annexe A

Spécification du fichier de configuration de la politique d'injection

Pour permettre la réutilisation d'une politique d'injection, celle-ci doit être sauvegardée, on a choisi alors d'utiliser un fichier de configuration. Et pour permettre une utilisation générique et extensible du contrôleur, on a opté pour une représentation XML de ce fichier, on a choisi l'extension « .ctl » signifiant « control ». Le fichier contient les informations suivantes :

A.1 Variables

Pour que le fichier « ctl » soit lisible et qu'il n'y ait pas beaucoup de répétitions, on a introduit des variables pour lesquelles on spécifie un attribut « name » pour indiquer le nom. Les variables sont de 2 types :

1. Des variables liées à Clif qui sont déclarées dans la balise « declareClifVariables », ces variables sont elles mêmes de 2 types :
   – Des variables en lecture seule qui sont définies dans la balise « statVars », une variable de ce type représente une mesure particulière correspondant à une lame. On précise pour ces variables le nom de la lame et l'indice de cette mesure dans la liste des mesures (cet ordre est indiqué dans les spécifications de Clif et peut être déduit simplement à partir du monitoring qui s'affiche dans la console). Par exemple, pour les sondes CPU, on a comme mesure, le pourcentage de CPU globale consommé, le pourcentage de CPU kernel consommée et le pourcentage de CPU user consommée.
   – Des variables en lecture et écriture qui sont définies dans « paramVars », une variable de ce type représente un des paramètres d'une lame. On précise pour ces variables le nom de la lame et le nom du paramètre, en spécifiant pour les paramètres de type population des injecteurs, le comportement concerné.
2. Des variables auxiliaires qui sont introduites pour faciliter l'écriture de l'algorithme et qui ne sont pas déclarées. On peut par exemple définir une variable (grandPas = 10)

et une variable (petitPas =2). La variable grandPas constituera l'incrément de charge si l'on considère que le système se stabilise vite (par exemple si le temp de stabilisation est inférieur à 10 s), et la variable petitPas constituera l'incrément de charge dans le cas contraire.

A.2 Expressions

Une expression peut être :

Une expression simple : scalaire (balise « scalar »), la valeur d'une variable (balise « variable ») ou bien la somme, le produit, la différence ou le résultat de la division de deux expressions. Dans ce cas L'expression est représentée par une balise dont le nom est celui de l'opérateur qui contient deux balises expressions représentant les opérandes.

A.3 Instructions

Puisque les instructions de contrôle se divisent en des instructions d'initialisation, et des instructions exécutées à chaque itération, le fichier contient 2 balises appelées respectivement « initInstructions » et « loopInstructions » qui contiennent des instructions.

L'instruction fondamentale, qui représente un ordre de modification d'un paramètre d'une lame, est mise dans une balise « changeParamInstr » en précisant la variable concernée (qui doit être du type « paramVar ») et la nouvelle valeur. De plus, on a mis les instructions d'injection dans des balises à part nommées « injectInstr », la valeur à injectée pouvant être précisée comme un attribut ou une balise expression.

Pour manipuler les variables, une instruction d'affectation de variables a été introduite sous forme d'une balise « affectVarInstr » qui contient un attribut «varName » spécifiant le nom de la variable et une balise de type expression.

On dispose aussi d'une instruction qui représente le conditionnement sous forme d'une balise « if » qui contient elle-même une balise condition, une balise représentant l'instruction exécutée si la condition est vérifiée appelée « then » et une instruction exécutée au cas contraire appelée « else ». La balise condition, contient une balise représentant une expression, un attribut représentant le comparateur et un attribut représentant la valeur à la quelle on veut comparer l'expression. Configuration

Quelques paramètres par défaut sont indiqués au début du fichier sous forme d'attributs d'une balise « config », comme la durée utile d'un palier, la durée d'une montée en charge, etc. Ces paramètres sont accessibles en lecture et écriture dans les instructions.

Pour permettre l'édition des fichiers « ctl », on a implémenté un nouveau Wizard dans la perspective CLIF qui permet entre autres de vérifier l'extension donnée. En utilisant la technique de développement de plugin sous Eclipse, on a implémenté aussi un éditeur minimal pour ce type de fichier, basé sur un éditeur de fichier XML qui offre un onglet d'édition textuelle et un onglet d'édition graphique sous forme d'un arbre. Par contre, cette interface n'est pas assez riche comme prévu. Ceci peut poser problème pour un utilisateur débutant qui n'est pas habitué à l'écriture de fichiers XML, nous avons alors décidé de prévoir des fichiers « ctl » simple pour ce genre d'utilisateurs. Dans ce type de fichiers, l'utilisateur aura uniquement à préciser une ressource critique (un nom d'une sonde de mesures) et le nom de l'injecteur dont il veut contrôler les injections.

A.4 Exemple

```
<control>
<config> <!--default values-->
      StabilisationDuration_S="2"
      RisingDuration_S="30"
      calculDuration_S="60"
      expectedSaturationLoad="100"
</config>

<declareClifVariables> <!-- declare variables in relation with Clif-->
      <statVars> <!-- read Only -->
         <statVar name="CPUConsumption" bladeName="CPUG-shiva2"
index="0"/>
      </statVars>
      <paramVars> <!-- read and write -->
         <paramVar name="injectorLoad" bladeName="InjectorG-load3"
param="population" behavior="model_Idf"/>
      </paramVars>
</declareClifVariables>

<initInstructions>           <!--initialisation instructions -->
 <injectInstr varName="injectorLoad" increment="1" /> <!--inject 1 Vuser--
>
</initInstructions>


<loopInstructions>
      <affectVarInstr varName="incrementation">
            <divide>
                  <minus>
                        <variable varName="expectedSaturationLoad"/>
                        <variable varName="injectorLoad"/>
                  </minus>
                  <scalar value="2"/>
            </divide>
      </affectVarInstr>
      <if>
            <condition comp="inf=" value="95">
                  <variable varName="CPUConsumption"/>
            </condition>
            <then>
               <if>
                  <condition comp="sup=" value="10">
                        <variable varName="incrementation"/>
                  </condition>
                  <then>
                        <affectVarInstr varName="incrementation">
                           <scalar value="10"/>
                        </affectVarInstr>
                  </then>
                  <else>
                        <affectVarInstr varName="incrementation">
                           <scalar value="2"/>
                        </affectVarInstr>
                  </else>
               </if>
               <injectInstr varName="injectorLoad">
                  <variable varName="incrementation"/>
               </injectInstr>
            </then>
      </if>
</loopInstructions>
</control>
```

FIGURE A.1 – Exemple de fichier de controle CTL

Résumé :

Les systèmes répartis sont caractérisés par une complexité croissante aux niveaux de l'architecture, des fonctionnalités et de la charge soumise. Dans le cas où la charge est importante, cette complexité induit souvent à une perte de la qualité de service offerte, ou une saturation des ressources, voire même l'indisponibilité des services en ligne. Afin d'éviter les problèmes causés par d'importantes charges et d'assurer le niveau de la qualité de service attendue, les systèmes nécessitent une auto-adaptation, en optimisant par exemple un tier ou en renforçant sa capacité par la répliquation. Cette propriété autonome requiert une modélisation des performances de ces systèmes.
Cette thèse présente une méthodologie théorique et expérimentale d'identification automatique de modèle et de dimensionnement des systèmes répartis. Après avoir décomposé un système réparti en un ensemble de composants boîtes noires, notre approche permet de déterminer les limites de chaque boîte noire et de la modéliser automatiquement par un modèle de file d'attente. Le processus d'identification du modèle se base sur des tests d'injection de charge auto-régulés et des tests d'hypothèse statistiques. Ensuite, les modèles des boites noires générés sont composés pour former les réseaux de file d'attente associés à chaque configuration candidate permettant le dimensionnement du système global.
Enfin, nous avons développé un framework générique, FAMI, implémentant notre méthodologie de modélisation automatique. Ce framwork a été couplé à des outils de simulation et d'analyse de réseaux de files d'attente permettant la résolution du modèle du système global et son dimensionnement.

Mots clés : Dimensionnement, autonomic computing, évaluation des performances , système répartis.

Abstract :

Modern distributed systems are characterized by a growing complexity at the level of their architecture, functionalities and workload. For important workload, this complexity leads to quality of service loss, saturation and unavailability of on-line services. To avoid troubles caused by important workloads and fulfill a given level of quality of service, systems need to self-manage, for instance by tuning or strengthening one tier through replication. This autonomic feature requires performance modeling of systems.
This thesis presents a theoretical and experimental methodology for the automatic model identification and sizing of distributed systems. After decomposing the system into a set of black boxes components, our approach allows to determine, first, the performance limits of each black box and models it automatically by a queue model. The identification process of the model is based on tests with self-regulated load injection and statistical hypothesis test. Second, the black boxes models are composed to form a queuing network model for each candidate configuration and enable the sizing of the global system.
Finally, to implement our methodology for automatic modeling we have developed a generic framework, FAMI. This framework was coupled with queuing network simulation and analysis tools to determine the performance of global model to size the system using analysis or simulation approach.

Keywords : Sizing, autonomic computing, Performance evaluation, distributed system.